University of Zurich[UZH]

Department of Informatics

Monetization strategies for the Web of Data

Dissertation submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Tobias Peter Grubenmann
from Appenzell, AI

approved in September 2018 at the request of

Prof. Dr. Abraham Bernstein
Prof. Dr. Elena Simperl

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, September 19, 2018

The Chairman of the Doctoral Board: Prof. Dr. Sven Seuken

Abstract

The World Wide Web knows two main monetization strategies to finance its content: advertisement and donations. Unfortunately, these strategies do not translate to the Web of Data because its machine-processable nature renders such approaches unpractical. Hence, new strategies are needed if we want the Web of Data to continue to grow. In this thesis, we present three new monetization strategies for the Web of Data.

The first strategy is a marketplace for data. Users are able to buy data in this marketplace in an integrated way, which means that they can access the complete data offering from all sellers as one transparent service. However, the users do not have to pay for the whole data offering for any given query. Instead, they only pay for the data that is required to produce the specific query answer for the given query. To achieve this, the query has to be executed before a buying decision is made by the user. To facilitate this buying decision, the user receives a summary of all the possible solutions for the query. As choosing the best set of solutions is an NP-hard problem, we propose an algorithm which solves the problem approximately in much shorter time.

The second strategy is an auction which allows third parties to promote data. Similar to sponsored-search auctions in the WWW, solutions containing URIs receiving higher bids get prioritized over other solutions. In contrast to sponsored-search auctions, however, the prioritization is achieved by delaying the delivery of less prioritized data. Typically, sponsors bid on URIs which redirect the users to a service. If a user looks up a URI, the corresponding bidder pays a price determined by the weighted VCG mechanism. For this purpose, we also introduce a new click-model, which is specifically designed for the WoD setting.

The third monetization strategy provides a way for consumers of streaming data to share the prices if their needs are overlapping. The more queries are participating in our system, the higher is the potential for price sharing and hence, saving money. We propose an algorithm to calculate the price shares and study how this algorithm meets the specific requirements that we elicited for this setting.

Using one of the first two strategies, data providers can monetize their wealth of data, depending on whether their data is more suitable to be sold directly or promoted by third parties. The third strategy focuses more on the user's ability to save money by exploiting synergies. With our work, we present new ways to build a financial sustainable Web of Data.

Zusammenfassung

Das World Wide Web kennt hauptsächlich zwei Strategien zur Finanzierung dessen Angebots: Werbung und Geldspenden. Leider lassen sich diese Strategien nicht direkt auf das Web of Data anwenden, da aufgrund des Fokus auf maschinelle Verarbeitung solche Ansätze nicht praktikabel sind. Deshalb sind neue Strategien für das Web of Data gefordert, um dessen Wachstum zu fördern. In dieser Dissertation stellen wir drei neue Strategien zur Finanzierung des Web of Data vor.

Die erste Strategie ist ein Marktplatz für Daten. Benutzer können in diesem Markplatz Daten von verschiedenen Anbietern kaufen als würden sie von einem zentralen Anbieter offeriert. Dazu müssen sie aber nicht für alle Daten bezahlen, welche im Marktplatz angebotenen werden. Stattdessen können die Benutzer nur diejenigen Daten kaufen, welche auch tatsächlich zur Beantwortung eines bestimmten Query benötigt werden. Dazu wird der Query auf allen Daten ausgeführt, bevor sich der Benutzer entscheidet welchen Teil der Daten zu kaufen. Um die Entscheidung zu erleichtern, erhält der Benutzer vom Markplatz eine Zusammenfassung aller verfügbaren Antworten zu einem bestimmten Query. Da eine optimale Kaufentscheidung ein NP-schweres Problem ist, bieten wir einen alternativen Algorithmus an, welcher das Problem näherungsweise in viel kürzerer Zeit lösen kann.

Die zweite Strategie ist eine Auktion welche es Dritten erlaubt, bestimmte Daten zu fördern. Ähnlich zu Auktionen für gesponserte Suchresultate im WWW werden hier solche Resultate bevorzugt die URIs mit höheren Geboten enthalten. Anders als bei Auktionen für gesponserte Suchresultate wird eine Bevorzugung durch das verzögern anderer Daten erreicht. Typischerweise werden Gebote auf solche URIs abgegeben welche den Benutzer auf einen bestimmten Dienst verweisen. Wenn ein Benutzer solch einer URI folgt, muss der entsprechende Sponsor einen Betrag bezahlen welcher nach dem gewichteten VCG Mechanismus berechnet wird. Zu diesem Zweck führen wir auch ein neues Klick-Modell ein welches spezifisch für das WoD entwickelt wurde.

Die dritte Strategie bietet für Konsumenten von Streaming Daten einen Weg um Kosten zu sparen, sollten sich ihre Bedürfnisse überschneiden. Je mehr Queries sich an unserem System beteiligen, desto grösser ist das Sparpotenzial für die Kunden. Wir schlagen einen Algorithmus zur Berechnung der Kostenanteile vor und untersuchen, inwiefern der Algorithmus den von uns festgelegten Anforderung nachkommt.

Datenanbieter können mit einem der ersten zwei vorgestellten Strategien ihren Reichtum an Daten finanzieren. Die Wahl der Strategie hängt dabei davon ab, ob sich die Daten eher für den Direktverkauf oder die Förderung durch Dritte eignet. Die dritte Strategie konzentriert sich mehr auf den Konsumenten und wie er Synergien nutzen kann um Geld zu sparen. Mit unserer Arbeit bieten wir neue Wege an um ein finanziell nachhaltiges Web of Data zu bilden.

Acknowledgements

First, I want to express my sincere gratitude to my supervisor Abraham Bernstein and my co-supervisor Sven Seuken for the opportunity of allowing me to pursue a Ph.D. at the University of Zurich. Without their continuous support and guidance, this thesis would not have been possible. Thanks also to Elena Simperl for taking the time to act as a co-referee for this work.

I also want to thank Dmitry Moor for his insightful feedback during our countless meetings. Another special thanks goes to Daniele Dell'Aglio, who helped me a lot during the last stage of my Ph.D.

During my time at the Dynamic and Distributed Information Systems group, I had the privilege to work with a lot of good friends and colleagues. Thanks to: Bibek, Shen, Cosmin, Daniel, Ela, Lorenz, Tom, Marc, Matthias, Michael, Patrick, Pengcheng, Philip, Christina, Suzanne, Malena, Narges, and Yasett.

Finally, thanks to my family and friends. Without their support, this thesis would not have been possible.

Financing

This work was partially supported by the Swiss National Science Foundation under grant #153598.

Table of Contents

III Appendix

Part I

Summary of this Thesis

Chapter 1

Synopsis

1.1 Introduction

In 1989, Tim Berners-Lee invented the World Wide Web (WWW) at CERN in Switzerland. The initial goal of the WWW was to facilitate information-sharing between scientists around the world. Four years later, in April 1993, the underlying technologies of the WWW became available to the public on a royalty-free basis. Since then, the WWW has experienced a phenomenal growth. [Cailliau, 1995]

A little bit more than ten years after the birth of the WWW, Tim Berners-Lee introduced the *Semantic Web*, also known as the *Web of Data* (WoD). In contrast to the WWW, the WoD focuses on the exchange and processing of *structured* data instead of documents. The idea of the WoD is to facilitate the processing of the data by machines. Thus, the WoD offers new ways of processing data autonomously. [Berners-Lee et al., 2001]

One big difference between the WWW and the WoD is how the content is accessed. In the WWW, users usually request single documents and consume them one by one. Following the links contained in the documents, users can find and explore other documents and, potentially, the whole WWW. The WoD does not have the concept of single documents. Instead, all data in the WoD form one global, decentralized graph. As a result, accessing content in the WoD is more similar to querying a federated database than browsing web pages. This fact has some serious implications on the monetization aspect of the WoD: most monetization strategies from the WWW cannot be applied easily to the WoD. *Advertisement* and *donations* are amongst the most important strategies in the WWW.

Nowadays, a lot of content in the WWW is financed through *advertisement*. By *exposing* users to advertisement, content providers in the WWW can generate a lot of revenue. Depending on the payment model, the advertiser pays money when an ad is shown to a user, when the user clicks on the ad, or when the user performs some action which is related to the ad (e.g., buying a product which was promoted). Both the space on a webpage and the users attention is limited, which creates high competition between advertisers and drives the prices up. As a result, advertisement in the WWW is a very lucrative business and, hence, creates strong incentives for data providers to publish data.

Given the big success of advertisement in the WWW, it is only natural to ask whether these mechanics could also be applied to the WoD. Unfortunately, this is not as straightforward as one might think. The problem is that the query language SPARQL [Harris and Seaborne, 2013], which is used to query content in the WoD, does not provide support for embedding ads. Even if SPARQL would support embedding ads, since a SPARQL query answer consists of structured data and is processable by machines, such advertisements

could easily be filtered out by an algorithm before the query answer is shown to the user. Even worse, the machine-processable nature of the WoD creates a lot of new use-cases, where no direct human interaction with the data is needed anymore and hence, there is nobody to see the advertisement. This renders advertisement not very effective in these use-cases.

Besides advertisement, some content provider in the WWW rely on *donations* as a main income of money. Donations usually rely on a group of loyal users who use the specific web content on a regular basis and have an interest in keeping the offered service alive. *Awareness* is a crucial component when financing a web service through donations. A user needs to be aware of the need for financial support. Wikipedia[1] is an example of a webpage which is exclusively funded through donations. As Glott et al. point out, the majority of visitors of Wikipedia either do not know how to donate or are not even aware that the website is in need for donations [Glott et al., 2010]. For this reason, a lot of data publishers relying on donations put banners on their websites to make the users aware of the possibility to donate.

In the WoD, the problems are even more pronounced. Embedding calls for donations into SPARQL query answers would not be very efficient, because the machines processing the data would simply ignore them. And without calls for donations, a user might not be aware that a data provider needs financial support. Even worse, a user might query a federation of different data providers and might not even be aware which data providers are supplying the necessary data. From the user's perspective, all data is part of one global graph and single data providers might not be perceived as such. As a result, most data providers might not be able to create the awareness to trigger donations.

While the WoD has the aforementioned drawbacks for advertisement and donations, it also has its advantages. Since all data sources are integrated into one big global knowledge graph, queries from different users can be executed on the same infrastructure. This especially opens new opportunities for stream queries over RDF data. RDF Stream Processing (RSP) introduces the dimension of time into query processing [Dell'Aglio et al., 2017]. As a result, RSP queries might have overlapping computation with other queries during their lifetime. This opens up opportunities for collaboration between different users to share the price of common computation.

There has been a lot of research on how to process and use data in the WoD, however, *monetization aspects have been ignored by the majority of research in this field.* Without new monetization strategies, many promising datasets will be poorly maintained or disappear as there will be not enough funds to keep the data up-to-date and the underlying servers running. To fill this gap, this thesis focuses on monetization strategies for the WoD. The reason why such new monetization strategies have to be studied is that, as discussed above, the WoD differs from the WWW in some important respects which offers new opportunities for monetization strategies but also renders some well-established strategies, such as advertisement and donations, useless.

In particular, this thesis focuses on three topics: (1) how data publishers can sell their data in a marketplace for the WoD, (2) how sponsors can finance data which they wish

[1] http://www.wikipedia.org

to promote using a special kind of auction, and (3) how users can save money by sharing the processing fees for streaming data. Each of these three topics covers one monetization aspect of the WoD which have been overlooked so far.

1.2 Background

This section introduces briefly some background about Semantic Web technologies, the economics of data, and RDF Stream Processing.

1.2.1 Semantic Web Technologies

In the Web of Data, the Resource Description Framework (RDF) [Cyganiak et al., 2014] is used to model relations between nodes in a directed, labeled graph, also called a *knowledge graph*. A node in RDF can be either a *resource*, a *literal*, or a *blank node*. A resource can denote anything, e.g., a website, an image, a physical thing, or an abstract entity. A literal is a string of characters with an optional datatype associated to it. Blank nodes are resources which do not possess an identifier. They can be used to refer to things without having to name them explicitly[2]. Each resource has an *identifier* which is a string that conforms to a specific syntax. The Web of Data knows two different kinds of identifiers, *Uniform Resource Identifiers* (URIs) and *Internationalized Resource Identifiers* (IRIs). IRIs are a generalization of URIs which allow a wider range of Unicode characters. In this thesis, most identifiers are for illustration purposes only. In these cases, more simple and compact identifiers are used to increase readability of the illustrations.

A *statement* in RDF is a 3-tuple of a subject, an object, and a predicate. The predicate defines the relation between the subject and object. RDF statements are also called *triples*. Throughout this thesis, RDF data is always illustrated in the same way, as depicted in Figure 1.1. The ovals represent RDF nodes, which act as subjects and objects of the statements. The arrows represent the predicates, which are pointing from the subject to the respective object of the statement. Nodes can be the subjects and objects of many different statements. Labels with quotation marks indicate literals and labels without quotation marks indicate resource identifiers.

Users (or machines) interested in the data represented in an RDF graph can use the SPARQL query language [Harris and Seaborne, 2013] to access it. Listing 1.1 shows an example of a SPARQL query. In this query, hotels and images are requested with the constraint that the hotel's name should be "Hotel California". Lines ending with a "." are called *triple patterns*. A triple pattern is like an RDF statement, but subject, predicate, or object can be replaced by a variable. Variables are indicated by identifiers starting with a "?". Triple patterns are organized into *Basic Graph Patterns* (BGPs), indicated by the enclosing curly brackets. A BGP is a set of triple patterns and can be used to match a whole subgraph of an RDF graph. Operators like filter expressions, joins, and unions can be applied to BGPs.

A SPARQL query consists of a *query pattern* and a *query form*. The query pattern of a SPARQL query is indicated by the WHERE clause. A *solution mapping* (or just *solution*)

[2] Blank nodes are not discussed in this thesis and hence, we are not considering them further.

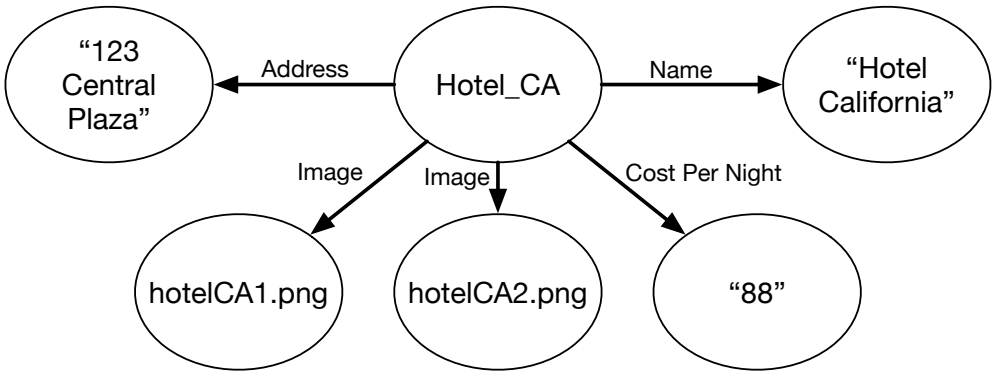


Fig. 1.1: An example RDF graph describing the resource Hotel_CA.

Listing 1.1: A query which asks for hotels named “Hotel California” and their images.

```
SELECT ?hotel ?image
WHERE {
  ?hotel Image ?image .
  ?hotel Name "Hotel California" .
}
```

is a set of bindings of the variables in the query pattern which matches the queried RDF graph. Table 1.1 shows an example of two solutions (one per row) for the query in Listing 1.1. The query form uses the solutions of the query pattern to build the final *query answer*. In this thesis, we will mainly focus on the SELECT query form. This query form allows its user to project the solutions to a subset of the variables that are used in the query pattern. If not specified otherwise, SPARQL queries are always considered to be SELECT queries. A *SPARQL endpoint* is a web address which accepts SPARQL queries and returns query answers based on the RDF graph stored in the back-end. In this thesis, SPARQL endpoints are also called *sources*. It is possible to query the union of multiple RDF graphs that are distributed over multiple SPARQL endpoints. For this, a SPARQL query must be split up into subqueries that are submitted to the different SPARQL endpoints. The individual query answers are joined and returned to the user. We call queries which are executed in such a distributed fashion *federated SPARQL queries*.

Table 1.1: Result of the query in Listing 1.1 over the graph in Figure 1.1.

?hotel	?image
Hotel_CA	hotelCA1.png
Hotel_CA	hotelCA2.png

1.2.2 Economics of Data

This thesis focuses on data as a digital good. Unlike physical goods, digital goods can be replicated at negligible costs and, as a consequence, the supply is (theoretically) infinite. This has some important consequences for selling data as a digital good. Assuming that two different data providers sell the exact the same data (and there is no difference in the quality of service), the cheaper of the two providers should be preferred by the users. But since there is no shortage of supply, the cheaper provider can serve all customers and hence, the second provider will not be able to sell anything. This specific situation is described by a model called *Bertrand Competition*, which describes what happens when two sellers simultaneously set prices for their undistinguishable goods and buyers want to buy from the seller with the cheapest price [Bertrand, 1883]. According to this model, prices will drop down to marginal costs. However, while creating data usually has high fixed costs, the marginal costs are (close to) zero. This puts the sellers in a very unfortunate position. Hence, we assume in this thesis that two different data suppliers do not possess the exact same data. While this seems like a very strong assumption, it is quite common for digital goods to be monopolistic: digital goods like music, books, or movies have a high degree of horizontal product differentiation and usually do not compete with each other directly [Peitz and Waldfogel, 2012]. However, we do consider the fact that there might be data which are substitutes for each other.

Another central concept of this thesis is the user's *value* for a query answer. The value indicates the maximal price the user is willing to pay. The user's value is specific for each query answer and can depend on various properties like quality and cardinality of the query answer. If not specified otherwise, we assume that the value is a linear function with respect to the solutions it contains.

Every query answer has a *price*, which can be determined using various different pricing mechanisms. If the price for a query answer is larger than the user's value, he or she will not buy this query answer. Given two query answers with the same value, the user will prefer the one with the lower price. In general, we assume that a user will always prefer the query answer with the largest *difference* between value and price. We will call this difference the *utility* of the user.

Whilst the user is usually interested in maximizing utility, economists are often interested in maximizing the *social welfare*. Social welfare is the total value generated by a system. The *costs* which occur in the system are expressed as negative value and are also considered in the social welfare. However, since the marginal costs are (almost) zero for digital goods such as data, the social welfare is given just by the value of data which is delivered to the customer minus the fixed costs. Since the fixed costs do not depend on how much data is sold, a system like a market works optimally—from a social welfare point of view—if all requested data is delivered to the customers. However, the sellers of data will set a price to maximize their revenue. For some customer, the price for the data might be higher than the customer's value and hence, not every customer will be served with all the requested data. Consequently, social welfare is not always maximized in such data markets.

To study different opportunities for data publishers in the WoD, we distinguish between two different kinds of data: *commercial data* and *sponsored data*. Each kind leads to a different monetization strategy.

The main difference between commercial data and sponsored data is whether somebody has an interest in *exposing* a user to the data or not. For commercial data, the user has to pay for the right to see the data. The data is seen as a commodity and the seller tries to make as much profit as possible with its data. For this, the seller sets a price which maximizes the expected revenue. Usually, not all users will be able to afford the asked price and hence, some users will not get the data they desire. Examples of commercial data are data about consumer behaviors and stock exchange data.

For sponsored data, the situation is different. The goal of a sponsor is to expose as much users as possible to some specific data. For this purpose, the sponsor is willing to pay money, which can be used to invest into the publishers of the data. Examples of sponsored data are information about hotels, restaurants, and data about articles for sale.

1.2.3 RDF Stream Processing

In Section 1.2.1, we discussed the processing of static RDF data. In a static setting, a user submits a SPARQL query and receives the query answer after the query has been processed. In contrast, RDF streaming data is processed continuously over some user-defined period. For this, there are various extensions to the SPARQL query language which support the continuous evaluation of queries over RDF streaming data. Such a streaming query can be decomposed into different parts which perform some computation relevant for the query (e.g., joining or aggregating data). Decomposing a query like this leads to a topology for the query where different computation nodes take one or several streams of data as an input, process the data, and produce another stream as output. If multiple queries are decomposed in this way, some of them might contain the same nodes (i.e. same inputs, same computation, and hence, also same output). Whenever two queries have the same node in their topology, the same process can be used for both queries and hence, computational power can be saved.

Figure 1.2 shows an example topology of three different queries, Q1, Q2, and Q3. The arrows indicate the flow of the streaming data. Note that the output of some of the computations are used for different queries. The consumers with the different queries will share the costs for the computation and source nodes they share with each other.

1.3 Problem Statement

As discussed in Section 1.1, the WoD suffers from a lack of monetization strategies as strategies from the WWW do not translate easily to this new setting and new opportunities in the streaming setting are not yet exploited. The focus of this thesis focuses are the following three topics:

- **Focus 1: How can data publishers sell commercial data in the WoD?**
- **Focus 2: How can sponsors promote sponsored data in the WoD?**

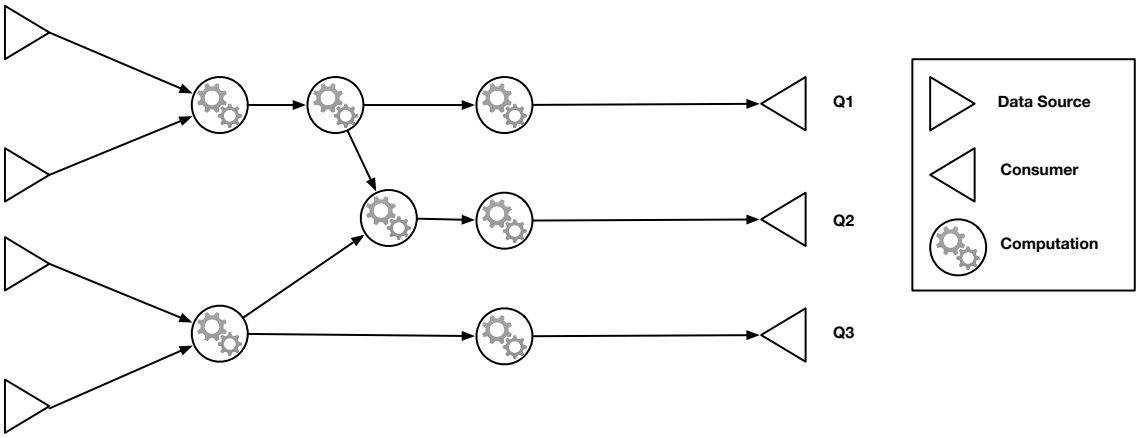


Fig. 1.2: Three different consumers with Queries Q1, Q2, and Q3 are sharing computation and source nodes.

– **Focus 3: How can users share prices for streaming data in the WoD?**

Each of these topics faces some unique challenges which leads to a research question and one or more hypotheses.

For the first focus, commercial data, the simplest solution would be that each data provider charges the consumer independently for its data offering. We identified the following challenge, which render such an approach unfavorable and hence, ask for better solutions:

Challenge 1: Joining sources: If a user wants to *join* data from multiple providers, a big part of the accessed data might get "lost", i.e., the data does not join and hence, is not part of the final query answer. Indeed, the user can use a left join to retrieve also the data which is accessed but not joined. But this would mean receiving a lot of data which is not of interest and, probably, of no value. Since a data provider cannot judge whether a user has value for the accessed data or not, the provider must charge for all accessed data, not just for data which ends up in the user's query answer, eventually. But such prices based on all accessed data might be too high for certain users, especially if only a tiny fraction of the data is required to answer the query. If there are a lot of sources that offer data, a user might not be able to judge ex-ante whether including a certain source into query execution justifies the specific ask price. In the worst case, choosing the wrong sources might result in a very low value for the user while paying a high price.

To allow users to make good decisions about which data to buy, they would require an accurate estimate of the cardinality of the query answer when executed on a specific set of solutions. As we will see in Section 1.5.1, there are limitations to the extend to which such accurate estimates can be made. Hence, we propose a marketplace for the WoD where the query is executed before the user has to decide which data to buy. However, this leads us to another challenge:

Challenge 2: The allocation problem: Even with complete knowledge about how much each different source can contribute to a specific query, selecting the best, meaning utility maximizing, combination of data is an NP-hard problem, which we call the *allocation problem*. As we show in this thesis, the allocation problem can be modelled as a variation of the knapsack problem and might require exponential runtime to solve. Without specific tools for this problem, users might be overwhelmed by the data offerings, which can be a big entry barrier when buying data in the WoD.

For the second focus, sponsored data, the WoD needs a way to incentivize potential data sponsors to pay money to promote data to consumers. For this, we identified the following challenge:
Challenge 3: Incentives for Sponsors: In principle, every data sponsor has an incentive to keep the service running that offers the data of interest. Different sponsors might have the interest in sponsoring the same service, as the offered data helps both sponsors to promote their data. The problem is that it is hard to incentivize a sponsor to invest money into a service which will also benefit other sponsors, who might not invest at all and might even be competitors. Hence, there must be some kind of benefit for those sponsors investing a lot of money into a certain service offering the data. It becomes natural to think about applying techniques from sponsored search auctions in the WWW to the WoD. However, this is not as straightforward as one would hope. A query answer is a set of solutions. Naturally, such a set does not possess any ordering of the elements. However, current mechanisms for sponsored search auctions are based on the fact that a higher rank in an ordered search result page has a higher value for the sponsor. In SPARQL, the requested data can be ordered ad libitum, which means that no specific ranking on the solutions can be imposed on the consumer of the data and hence, there is no competition to obtain a higher rank.

Finally, the last focus is about streaming data. RDF Stream Processing could be much more attractive for consumers if the prices could be shared should their needs overlap. However, there is also a challenge involved in realizing such price sharing:
Challenge 4: Manipulating price shares: Opportunities for saving money often lead to opportunities of manipulating the underlying system by misreporting certain input. The problem with such misreports is that they often benefit the one misreporting at the expense of other participants. Hence, a price sharing algorithm has to make sure that the participants cannot benefit by misreporting. Such misreports include deviating from the original query or lie about the parameters (e.g. maximal lifespan and maximal amount the customer can spend) with which the query should be executed.

1.4 Research Questions and Hypotheses

This thesis addresses four different research questions related to the four challenges introduced in the last section. Each research question is addressed in one of the four different papers presented in Part II. Each research questions leads to one or more hypotheses.

The first two challenges are related to selling commercial data in the WoD. The first challenge is about the fact that a user might not know ex-ante how much data will appear in the query answer and whether this would justify the price. Hence, we ask in Section 1.4.1 whether it is possible to estimate the value of a query answer prior to executing the query and thus, paying for the data.

The limitations of estimating the cardinality of a query answer leads to our idea of a marketplace, where the query is executed before selling the data. However, this introduces a new challenge, the allocation problem. In Section 1.4.2, we show how to solve the allocation problem for the customer and hence, lower the entry barrier for costumers in our market.

After the discussion of the first two challenges, we switch the focus to sponsored data. In Challenge 3 we are facing the problem of creating incentives for sponsors to pay money. Our answer to this problem is the *delay-answer auction*. In Section 1.4.3, we asks ourselves how the choice of certain parameters of this auction will influence the participants.

Finally, we switch to the last focus of this thesis: streaming data. The challenge we identified here is that participants of a sharing algorithm might exploit the system to their advantage and at the costs of others. In Section 1.4.4, we ask about the requirements a price sharing algorithm should fulfil, with a special focus on misreports by participants.

1.4.1 Can one estimate a source's contribution accurately?

Most of the relevant data for a given SPARQL query is not located at a single endpoint but distributed over a lot of different sources. Querying all relevant data from the respective sources yields a *complete query answer*. A complete query answer contains all solutions which can be obtained from the available sources. Depending on the prices the different sources charge for their data, such a complete query answer might be very expensive for the customer. If some data is excluded from query execution (either by excluding entire sources or part of their data offering) some solutions of the complete query answer might get lost. At the same time, the price of the query answer might drop, which can result in an increase in utility. In order to select the most beneficial (i.e., utility maximizing) combination of sources, one has to estimate the utility of every combination of sources, given the query.

To estimate the utility of a query answer, estimates of the *value* and *price* are required. For a specific combination of sources, the price is dictated by the participating sources and can be just summed up. However, estimating the value of a query answer is not as straightforward. For this part of the thesis, we assume that the value is proportional to the *cardinality* of the query answer. However, the cardinality of a query answer obtained from a combination of sources cannot be directly derived from the cardinality of the data involved. This is because only a part of the data involved in query execution might join with data from other sources. Hence, one has to estimate the join-cardinality between the different sources. Such estimates can be obtained by executing the given query in an approximate fashion. This leads to the following research question:

Research Question 1: What are the challenges and limitations of join-cardinality approximations for federated SPARQL queries?

The answer to Research Question 1 will determine the applicability of join-cardinality approximations for buying data in the WoD. If such approximations are applicable, a customer can approximate a query and decide afterwards which sources should be included in the query execution for the final result. If they are not applicable, however, other alternatives for buying data in the WoD have to be considered.

Bloom Filters are an example of data synopses which are used in join-cardinality approximations. The first question we attempt to answer in this thesis is whether Bloom Filters are useful synopses in this setting. The problem with Bloom Filters, as with any synopsis bearing false positives, is that the error can quickly explode when having a lot of joins. This was already observed by [Ioannidis and Christodoulakis, 1991]. In addition, we hypothesize that the false positives have a negative impact on the runtime behavior of the approximation:

Hypothesis 1: Join-approximations using Bloom Filters run slower when using Bloom Filters having a higher false positive rate.

To explain the error behavior analytically, we extend the work of [Ioannidis and Christodoulakis, 1991]. Our hypothesis is that the error depends on two main factors, the false positive rate and the join-selectivity. The false positive rate indicates how many new data items are erroneously introduced at each join. The join-selectivity indicates how many data items are rightly introduced at each join. Hence, the ratio between the two will mainly determine the relative error of the approximation:

Hypothesis 2: The relative error of join-cardinality approximations is proportional to the ratio between the false positive rate of the used synopsis and the join-selectivity of the basic graph patterns.

1.4.2 How can we build a market for commercial data in the WoD?

The second research question is about building a market for commercial data. To circumvent Challenge 1, we let the market—essentially and intermediary—execute the query before the customer makes a buying decision. The market then submits a *summary*, which doesn't reveal the actual data, to the customer, who can then decide which data to buy. We call such a buying decision an *allocation*. This does not solve Challenge 2, however. This leads to my second research question:

Research Question 2: How to solve the allocation problem for a market for the WoD efficiently?

Due to the *combinatorial* nature of our market, finding an optimal allocation is a variation of the knapsack problem which belongs the class of Integer Programming problems. Hence, our next hypothesis is:

Hypothesis 3: The allocation problem for the market for commercial data can be modeled as an Integer Programming problem.

Since solving an Integer Programming problem is NP-hard, we introduce an algorithm which can approximate an optimal solution having a more favorable time complexity.

1.4.3 How can we build an auction for sponsored data in the WoD?

As mentioned in Section 1.3, a sponsor has an incentive to invest in any data which increases the chance that a query answer contains the promoted data. However, it is important to give sponsors some kind of advantage over free riders. We solve this problem by introducing a *delay* to different solutions which are part of the query answer. The more money a sponsor is paying, the smaller the delay of the respective solution. To define the payment for a specific delay, we combine the weighted VCG auction [Nisan and Ronen, 2007] with a novel click-model which considers the different delays of the solutions. We call this new auction the *delayed-answer auction*. The different delays which can be assigned to different solutions are additional parameters which have to be set by the auctioneer before the auction begins. Our next research question asks about the influence of this parameters:

Research Question 3: How does the choice of the delays influence the proposed delayed-answer auction?

The first hypothesis is about how many different delay parameters should be chosen by the auction designer. How valuable a slot is does not only depend on the delay of the slot itself, but also on the delays of all other slots. If one would want to maximize the value of a given slot, one would have to make sure that: (1) the slot itself has minimal delay, making it the most favorable one, and (2) all subsequent slots have a maximal delay, making them the least favorable ones. The higher the difference between the most favorable and the least favorable slots are, the more money can be charged for the most favorable slots. A similar argument works for the social welfare. This leads us to the following hypothesis:

Hypothesis 4: The auctioneer's revenue and the social welfare are maximized when there are only two batches of solutions: one batch of the solutions is delivered immediately and the second batch is delivered as late as possible.

The next hypothesis asks whether it is possible to maximize the auctioneer's revenue and social welfare with the same set of parameters. We do not believe that this is the case because to maximize social welfare we want to maximize the expected value of the bidders but to maximize revenue we want to maximize the expected price the bidders have to pay. Hence, our hypothesis is:

Hypothesis 5: Delays which maximize the auctioneer's revenue do not maximize the social welfare, necessarily.

1.4.4 How can we share prices for RSP?

The last research question is about price sharing for RDF Stream Processing. As discussed in Section 1.3, the challenge in this setting is that participants are usually trying to manipulate a system if there is a chance of saving money. The last research question is about how to build a sustainable system, which is beneficial for the participants but does not allow the users to manipulate by misreporting:

Research Question 4: What are the requirements to ensure a sustainable cost sharing model when sharing costs in RSP?

In Chapter 5 of this thesis, we introduce an algorithm for RSP price sharing. We also introduce three requirements a sharing algorithm should fulfil in this setting.

The first requirement is that all participants should actually benefit from participating in our sharing model. Hence, the first hypothesis is:

Hypothesis 6: Each participant has a higher utility when participating in our sharing model.

Next, we require that no participant can benefit by misreporting the parameters needed to calculate the price shares. The problem with such benefits by misreporting is that they often are at the expense of other participants. We hypothesize that our algorithm does not allow such behavior:

Hypothesis 7: Participants in our sharing model cannot benefit by misreporting the required parameters.

Finally, deviating from the original query is another source of manipulation which might increase the utility of a participant at the expense of the others. We hypothesize that also this is not possible:

Hypothesis 8: Participants cannot benefit by deviating from their original query.

1.5 Contributions

This section summarizes the main findings regarding each research question. Details can be found in the respective papers in Part II.

1.5.1 Join-Cardinality Estimation for SPARQL Queries

To answer Research Question 1, we study the general error behavior of SPARQL join-approximations and evaluate how well Bloom Filters perform for the task.

In Chapter 2, we evaluate how Bloom Filters perform for the task of approximate query execution to estimate join-cardinalities. We use our own query approximation engine to estimate the cardinalities of the FedBench [Schmidt et al., 2011] queries using Bloom Filters as data synopsis. Our evaluation shows that the runtime of the approximations is a lot *slower* for many queries than the query execution on the actual data. This behavior renders the query approximation useless for those queries, as the purpose of a query approximation is to trade-in less accurate information for a reduced processing time. Surprisingly, the runtime of the approximation often *improves* when using less compressed (and more accurate) synopses. At the same time, the accuracy of cardinality estimation improves because of the more accurate synopses. The explanation for this counter-intuitive behavior lies in the fact that less accurate synopses also introduce more false positives into the approximation process. The overhead in processing the additional

false positives have a bigger negative impact on the runtime than the reduction of the size of the synopsis has a positive impact. The result partly verifies Hypothesis 1: for most queries, the approximation runs slower when using higher false positive probabilities.

Our analysis of the error shows that it is indeed proportional to the ratio between the false positive rate of the used synopsis and the join-selectivity of the basic graph patterns. Hence, for queries with low join-selectivity, the false positive rate also has to be very low, otherwise the error will explode. This verifies Hypothesis 2 and points to a general problem of approximating SPARQL queries: If the data is distributed over a lot of different sources, there are a lot of joins involved. In addition, the more joins are needed, the lower the selectivity of each join, because BGPs with less triple patterns are less specific and yield more solutions.

The main conclusion of this part of our research is that a customer is not able to estimate the cardinality and hence, the utility of a query answer accurately without executing the query. Therefore, we search of a way to fix this issue which leads us to the next contribution.

1.5.2 Selling Data in a Federated Fashion

For commercial data, we propose a marketplace in the WoD. The marketplace allows a customer to combine data from different sellers in an integrated way, which means that the customer can access all data from all sellers as if they were offered by a single entity. However, the customer does not have to pay for the whole data offering. Instead, a user only pays for those triples which are required to form the specific query answer for the given query. Since buying all solutions might be too expensive for the customer, he or she can decide how many solutions to buy.

This leads us to Research Question 2: how should the user decide in an efficient way which of all possible combination of solutions to buy? In Chapter 3, we show how this problem can be modeled as an Integer Programming problem. For this, the user's value has to be linear with respect to the solutions of the query answer. With the description of how to formulate the Integer Programming problem, we verify Hypothesis 3.

Our evaluation using FedBench [Schmidt et al., 2011] shows that solving this Integer Programming problem using CPLEX—a state-of-the-art solver for such problems—can take 10 to 30 seconds per query. While this might be still in an acceptable range, the runtime can be much worse when we change the *diversity*, which indicates how many solutions depend on the same triples. For certain diversity factors, the Integer Programming problem is not solvable anymore for a few hundred solutions within a time limit of 12 hours. This raises the question whether an algorithm approximating an optimal allocation can provide a satisfying allocation within a much shorter time frame. In Chapter 3, we present such an algorithm and compare it against CPLEX. We show that the algorithm has a time complexity of $O(n^2 \log(n))$. As expected, the algorithm performs much better with respect to the runtime. In most cases, our algorithm runs between 1 and 3 orders of magnitude faster than CPLEX. The allocation provided by our new algorithm is close to optimal. The algorithm reaches more than 90% of the utility of CPLEX in 15 out of 17 queries. For the other two queries, our algorithm reaches 85.0% and 79.8%.

1.5.3 Financing Data through Sponsors

To create incentives for sponsors to invest into the WoD, we propose to delay different part of a query answer. Each sponsor has the opportunity to place a bid on any resource which is part of the datasets participating in our system. Typically, such a resource redirects a user to a certain kind of service, if looked up. If the customer decides to look up a certain resource, the sponsor has to pay a payment which is calculated using the weighted VCG mechanism [Clarke, 1971, Groves, 1973, Vickrey, 1961]. The higher the placed bid on a certain resource, the smaller the delay of the solutions which contain the respective resource. Solutions with less delay have a higher chance of being considered by the user and hence, the probability that the user will look up the contained resource is higher.

The auctioneer must set the different delays as a parameter. The choice of these parameters influences both the generated revenue of the auction and the social welfare of the sponsors. In Chapter 4, we perform an analysis of our auction model and show that the auctioneer must set the delays in a way such that one part of the solutions is delivered immediately and the other part is delivered as late as possible, to maximize social welfare and revenue. This observation verifies Hypothesis 4. Unfortunately, it is in general not possible to optimize social welfare and revenue with the same set of parameters. As our simulation in Chapter 4 shows, the parameters which optimize social welfare can sometimes be very far from the parameters that optimize revenue and therefore, Hypothesis 5 is verified.

1.5.4 Saving Money by Price Sharing for Streaming Data

To make RDF Stream Processing more attractive, we propose an algorithm for price sharing. The algorithm identifies common computations among different queries. For each computation node, the algorithm distributes the costs of this node among all queries which require the node to produce its output stream. Each query receives an equal price share of the total cost of the computation node. Summing up all the price shares of all the required nodes for a specific query results in the total price. If the total price of a query is lower or equal than the user's value for the stream, the query gets some runtime allocated. There are two constraints influencing the allocated runtime: the budget and the maximal runtime. Whenever new queries are arriving in our model, new opportunities for price sharing might arise and hence, the price shares will be recalculated. As we show in Chapter 5, the runtime of the algorithm is mainly influenced by the number of queries. In its current implementation, our algorithm can serve more than 1000 queries simultaneously within a time limit of 10 seconds for calculating the price shares.

To ensure a sustainable model, we formulated three different requirements. The first requirement is that a user can actually benefit from the price sharing when participating. As we show in Chapter 5, our algorithm ensures that each participant receives a higher utility as long as there is at least some overlapping computation with other queries. This verifies Hypothesis 6.

The opportunity for price sharing also opens up opportunities for the participants to manipulate the model. In addition to the query, each user has to provide a value for the

requested stream, an upper bound for the execution time, and a budget. A user could try to manipulate the model by misreporting these parameters. Our analysis show that a user cannot actually benefit by misreporting and hence, Hypothesis 7 is also verified.

To verify Hypothesis 8, we have to show that a user cannot gain any benefit by deviating from the original query. As we discuss in Chapter 5, this is only true under certain conditions. First, we must assume that a user has no value for any partial stream of the requested stream. Second, we must assume that a user does not share the received stream with any third party. Hence, the hypothesis is only verified under these additional conditions.

1.6 Limitations

In this section, some of the limitations of the four contributions are discussed. Details of the limitations can be found in the limitations section in the different contributions in Part II.

In our research, we assume that the user has enough knowledge about the available datasets to formulate a query which requests exactly the desired data. In particular, we assume that a user has no need to explore the datasets before issuing a query. Without this assumption, we would have to provide means for the user to investigate the datasets before buying the data in FedMark, our prototype marketplace. This could be achieved by providing samples of the data, for example. However, additional care would be necessary to prevent users from exploiting such mechanisms. Also, our delayed-answer auction depends on the assumption that a user has enough knowledge to issue the correct query. If this assumption would not hold, the click model we must use would be much more complicated.

For FedMark, we also assume that the user's value is linear with the number of solutions in the query answer. However, we relaxed this assumption when introducing our own algorithm. In all cases, we assumed that the user has no value for an empty result. This assumption is based on the fact that a query can be empty, either, because there are no solutions to the query at all, or, because the datasets that could yield a solution are currently not participating in the market. Hence, an empty query answer is not always informative to the user. In addition to the assumptions about the value, we also assume that the utility equals the value minus the price. This assumption must not hold, necessarily. Other utility functions would be conceivable. Finally, we assume that the user has knowledge about his or her own value. Also, this assumption must not hold, necessarily. A user might struggle to determine the exact value of a yet unknown solution to a given SPARQL query.

For the sellers in FedMark, we assume that they already know how to price their data. In practice, the sellers have to learn how to optimize the price of the data, for example by using reinforcement learning. We also assume that the sellers are not resource bounded. This means that a seller can serve as much data as the different users are requesting, without reaching a limit on computing or network resources.

In our delayed-answer auction, we assume that solutions with higher delays have a lower probability of being selected by the user. This might not be true for all kind of cases

and hence, our work is limited to queries where such delays do matter. In addition, we assume that there is only one sponsor per solution. For a lot of queries this assumption might be true. However, if a user searches for specific combinations of services, there might be situations where there is more than one sponsor per solution. Our delayed-answer auction can be extended to settings with more than one sponsor per solution, however, in this case the VCG mechanism is not anymore applicable. Finally, we also did not discuss the distribution of the revenue among the data providers. This is left for future work.

One limitation of our price sharing algorithm for RDF Stream Processing is its scalability. As soon as tens of thousands of queries are simultaneously involved in price sharing, our algorithm might take some time to compute all price shares. Another limitation is the robustness of the algorithm against query deviations. This requirement is only met if we assume that the user has no value for a partial stream and the user does not share the received stream with third parties.

1.7 Conclusions and Future Work

This thesis studies monetization strategies for the WoD. The thesis consists of four different projects presented in Part II. The first two projects focus on commercial data. The first project discusses the challenges of selecting the most beneficial combination of sources for query execution. The second project offers a solution to the challenges encountered in project 1: FedMark. The third project focuses on sponsored data and presents our concept of a delayed-answer auction. Finally, the fourth project introduces our algorithm for RSP price sharing.

The first project set out to investigate the applicability of join approximations for source selection. We hypothesized that the performance of cardinality approximations of federated SPARQL queries degenerates when applied to queries with multiple joins of low selectivity. Indeed, both our empirical evaluation and our theoretical considerations indicate that data synopses are not suitable for this task due to their cumulative error, which also substantially slows down the estimation process. The consequence of our study is that a market for commercial data cannot rely on such estimates and hence, has to execute a customer's query on all datasets before a buying decision is made. Nevertheless, join approximations have their applicability in our new market concept. Estimates can be used to optimize the market workflow: a customer's query is first executed only on the most promising sources which have the highest potential of maximizing the utility. Executing the customer's query on less promising sources can be delayed giving priority to other queries. Such optimizations provide a trade-off between maximizing a single customer's utility and maximizing the number of queries that can be served within a given time-frame. Investigating such optimizations and their trade-offs is a task for future work.

In the second project, we presented FedMark, a prototype that implements the concepts we introduced for this setting. In addition, we presented two possible ways by which a customer can decide for a specific allocation. Using an Integer Programming solver guarantees an optimal solution, but the algorithm can take a lot of time to terminate. Our

own algorithm is usually much faster in finding an allocation but does not find an optimal allocation, in general. Another advantage of our own algorithm is that it can handle decreasing marginal values and hence, relax our initial limitations of only considering linear values. In future work, other algorithms can be developed and compared against our two initial algorithms. With our work, we established a first baseline on how such algorithms can be compared for the problem at hand. Another interesting topic for future work is the study of subscription-based models and the combination of such models with our current approach.

We also introduced a new concept of a delayed-answer auction to finance sponsored data in the WoD. We have shown that it is often not possible to find parameters which maximizes revenue and social welfare at the same time. It will be the choice of the auction designer to find a suitable trade-off between them. What is left for future work is the distribution of the generated revenue among the data providers. The revenue can be used to finance those providers which proved to be important for answering queries or subsidize those providers which struggle the most to keep their services running.

Finally, we proposed a sharing model for RDF Stream Processing. We showed that our algorithm can compute the price shares for 1000 queries in less than 10 seconds. To ensure a sustainable model, we also formulated three different requirements for price sharing. While our algorithm meets the first two requirements, the third requirement is met only under certain conditions.

Future research should focus on the following topics:

In this thesis, it is assumed that the sellers of data are interested in maximizing their profit. However, other incentive mechanisms could be used to motivate the creation and sharing of data. Users could be prioritized during query execution if they contributed in the past data to the global knowledge graph. Future research can investigate such alternative markets without money.

The distribution of data around the world is one big challenge for the Web of Data. So far, we have been focusing on how to collect and sell this distributed data. However, another interesting question is how the data should be distributed efficiently in the first place. Marketplaces could be used to organize the distribution of data. Finding an efficient data distribution while considering the supply and demand of storage is a challenging task for future work.

Inspired by the idea of sharing the costs of steam processing, one could also think about sharing costs for AI and Data Mining tasks on the Web of Data. By exploiting synergies, such tasks could be executed much more efficiently and at a lower cost for all participants.

A final task for future work will be to combine the monetization strategies introduced in this thesis into one integrated system. Ultimately, a customer should be able to combine commercial data, sponsored data, and streaming data.

In this thesis, we proposed different strategies to finance the WoD. Depending on the kind of data, providers can choose one of these monetization strategies: If the data is more suitable to be sold directly to the customer, providers can participate in a market like FedMark and make money with their data offering. In contrast, if there is some third

party which could act as a sponsor for the data, our delay-auction is the strategy of choice. Finally, if the data is streaming, the provider should think about integrating the data into our price sharing model, which makes streaming data more affordable for the customers and hence, can also increase the revenue of the data provider. With our work, we built the ground for these different strategies. We hope that our research helps data providers to secure the financing for their data offerings and thus, enable a more sustainable Web of Data.

Part II

Contributions of this Thesis

Chapter 2

Join-Cardinality Estimation for SPARQL Queries

This chapter is based on:

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. 2017. ***Challenges of Source Selection in the WoD****. In: d'Amato C. et al. (eds) The Semantic Web – ISWC 2017. Lecture Notes in Computer Science, vol 10587. Springer, Cham*

Challenges of source selection in the WoD

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken

Department of Informatics, University of Zurich, Switzerland

Abstract. Federated querying, the idea to execute queries over several distributed knowledge bases, lies at the core of the semantic web vision. To accommodate this vision, SPARQL provides the SERVICE keyword that allows one to allocate sub-queries to servers. In many cases, however, data may be available from multiple sources resulting in a combinatorially growing number of alternative allocations of subqueries to sources. Running a federated query on all possible sources might not be very lucrative from a user's point of view if extensive execution times or fees are involved in accessing the sources' data. To address this shortcoming, federated join-cardinality approximation techniques have been proposed to narrow down the number of possible allocations to a few most promising (or results-yielding) ones.
In this paper, we analyze the usefulness of cardinality approximation for source selection. We compare both the runtime and accuracy of Bloom Filters empirically and elaborate on their suitability and limitations for different kind of queries. As we show, the performance of cardinality approximations of federated SPARQL queries degenerates when applied to queries with multiple joins of low selectivity. We generalize our results analytically to any estimation technique exhibiting false positives. These findings argue for a renewed effort to find novel join-cardinality approximation techniques or a change of paradigm in query execution to settings, where such estimations play a less important role.

2.1 Introduction

At the core of the Semantic Web vision lies the possibility to ubiquitously access distributed, machine-readable, linked data. This Web of Data (WoD) relies on the notion of being able to access partial information from a variety of sources that then gets combined to an integrated answer.

One major approach to achieving this functionality in a distributed fashion is federated querying [Acosta et al., 2011, Basca and Bernstein, 2014, Buil-Aranda et al., 2013a, Kossmann, 2000, Ozsu and Valduriez, 1999, Schwarte et al., 2011a, Sheth and Larson, 1990]. It relies on traditional database approaches to join partial results from multiple sources into a combined answer. Specifically, it divides a query into subqueries and delegates the execution of each of these subqueries to one or more remote databases, which on the WoD are called endpoints. A query execution plan assigns the subqueries to a certain set of endpoints and determines the order of the subquery execution. Hereby, results from one subquery can vastly reduce the computational effort of answering another. One major problem of querying the Web of Data is *source selection*, which is deciding which subqueries should be delegated to which SPARQL endpoints during query execution and which endpoints should not be considered for query execution at all. We will focus in this paper on the cardinality of the query answer as the metric to evaluate the worthiness of including certain sources into query execution.

Ideally, a user would be able to estimate the cardinality of the query answer for any subset of all relevant sources. Given the knowledge about the resulting cardinality for different combinations of sources, a user could make an informed decision whether a certain

source should be included into the federated query execution or not. By an informed decision we mean deciding whether selecting and accessing a certain subset of all available endpoints is worth the time and, potentially, fees which are associated with accessing these endpoints.

In this paper, we argue that **the performance of *cardinality* approximations of federated SPARQL queries degenerates when applied to queries with multiple joins having low join selectivities**. This means that such approximations are not sufficiently precise to allow a user to make an informed decision. As a consequence, a user who cannot afford to query all relevant sources for a given query must blindly exclude some relevant sources risking low cardinality or empty query answers, even though solutions to the query would be available on the WoD. Specifically, our contributions are:

- We show empirically that the *cumulative error of cardinality estimation techniques based on Bloom Filters explodes* in the combinatorial distributed setting of the WoD, which questions its usefulness for informed source selection.
- We show empirically that the explosion of the *cumulative error often makes join-cardinality estimation slower than executing the actual query*. Hence, using such a technique may not only lead to suboptimal results but even slow down the query execution process, which is exactly the opposite of the goal of source selection.
- Using a theoretical analysis of the problem, we explain why these *negative results necessarily occur when using any estimation technique exhibiting false positives* in combination with queries having low join selectivities.

The remainder of this paper is organized as follows. First, we succinctly discuss the most relevant related work. Next, Section 2.3 provides empirical evidence of our claims about the limited usefulness of join approximation techniques using Bloom Filters, which is followed by a discussion of the results. In Section 2.4, we present our main result: a theoretical analysis which explains the cumulative error and associated runtime behavior that federated cardinality approximation techniques face in the WoD. We close with some conclusions.

2.2 Related Work

Federated SPARQL querying and source selection: Different approaches have been proposed to query RDF data in a federated setting. Mediator systems like FedX [Schwarte et al., 2011a] and DARQ [Quilitz and Leser, 2008] allow a user to query a federation of endpoints in a transparent way while incorporating all known SPARQL endpoints into the query answer. The federation appears to the user as one big SPARQL endpoint holding the data of all the members of the federation. Once the members are specified and initialized, the user can issue SPARQL queries against the mediator without having to adapt the query for federated execution or providing any additional information about the federation members.

Avalanche [Basca and Bernstein, 2014] and ANAPSID [Acosta et al., 2011] propose different, more dynamic systems where they relax the requirement of complete results

and allow certain endpoints to fail. Their systems focus on robustness of query execution in the Web. Avalanche [Basca and Bernstein, 2014] executes all possible queries (i.e., all combinations of possible endpoints) in parallel eventually timing out a query when the rate of incoming results slows down. In queries with many combinations this may lead to a very high network load and a significant time between querying and query completion. ANAPSID [Acosta et al., 2011], in contrast, runs only one query plan and dispatches each sub-query to every possible endpoint using a mediator. This results in a highly robust execution but again, faces the danger of including a very large number of endpoints if no sensible source-selection approach is available.

SPLENDID [Görlitz and Staab, 2011] proposed to exploit service descriptions and VoID statistics about each endpoint to perform source selection and query optimization. HiBISCuS [Saleem and Ngonga Ngomo, 2014] uses join-aware techniques to select relevant sources for federated query execution. HiBISCuS maintains an index which stores the authorities of certain URIs. [Vidal et al., 2016] introduced Fed-DSATUR, an algorithm for SPARQL query decomposition in federated settings. They do not use statistics, indices, or estimates for source selection.

The SPARQL 1.1 Federated Query extension [Harris and Seaborne, 2013] follows a different approach: a user must explicitly specify which part of the query should be executed on which server. The extension requires the user to know which SPARQL endpoint can provide data for which subquery and rewrite the query accordingly using a special SERVICE-clause.

Duplicate aware source selection [Saleem et al., 2013] tries to eliminate sources with duplicate data using Min-Wise Independent Permutations. [Hose and Schenkel, 2012] used Bloom Filters for source selection of RDF sources and investigated the number of requests needed for an approximation to achieve a certain recall.

Good estimates of the contribution of different sources towards a query answer plays an important role in [Moor et al., 2015] and [Moor et al., 2016], where users have to pay for accessing the selected sources.

In contrast to the work presented so far, we perform an empirical and theoretical analysis of the error behavior for the problem of source selection when the cardinality of the result is used as the deciding factor.

Cardinality Estimation Techniques:

In the traditional database domain, join approximation has been used as a suitable technique for approximate query processing [Chakrabarti et al., 2001]. The goal of approximate query processing is to compute an answer that approximates the query answer without having to execute the query. Join approximations can be used to calculate the expected cardinality and the join selectivity of a specific query.

A variety of approaches provide *data synopses* (i.e., summaries of the data) for join approximation. Histograms [Ioannidis and Poosala, 1999] and Wavelets [Chakrabarti et al., 2001] have been used to approximate the distribution of a dataset over a given domain. Also, Bloom Filters were first proposed as a space-efficient probabilistic data structure to approximate sets [Bloom, 1970]. The advantage of Bloom Filters is that they allow to specify the desired false-positive rate for set-membership checking without leading to

false-negatives. Given that they also allow intersections between bloom-filtered sets they have become a de-facto standard for join approximations. Q-Trees [Prasser et al., 2012] were introduced as a special data summary technique for RDF data. [Umbrich et al., 2011] compared the runtime and space complexity of indexing techniques, multidimensional histograms, and Q-Trees and evaluated, in particular, their usefulness for source selection and highlighted the superiority of Q-Trees over the others.
Sampling methods [Lipton et al., 1990] do not rely on a synopsis but on a selection of the data. Hence, they do not produce false positive matches but might produce false negatives. Sampling methods provide a lower bound on the cardinality of a join.

Join synopses [Acharya et al., 1999] are special summary structures built for join approximation. They are constructed for specific, ex-ante known join operations and are therefore not suitable to the purely ad-hoc federated settings. They are, however, useful when one knows that certain joins are likely to occur.

Finally, [Ioannidis and Christodoulakis, 1991] studied the propagation of errors in the size of the join result. In this paper, we will extend the analysis done by [Ioannidis and Christodoulakis, 1991] to the domain of SPARQL queries.

2.3 Experimental Evaluation of the Cumulative Join Estimation Error

The goal of this section is to show the relative error and runtime behavior of join cardinality approximation using Bloom Filters, which motivated our theoretical analysis of the problem and our conclusion that join approximation techniques are problematic for source selection. We used Bloom Filters for the approximation as they provide an easy and straightforward way to encode strings like IRIs and Literals.

In the following, we will first describe the experimental setup, including the query approximation engine and the data we used before presenting the results.

2.3.1 Query Approximation Engine

We implemented a query engine that allows us to execute joins over federated SPARQL endpoints on dynamically generated data synopses. The query engine accepts a query consisting of basic graph patterns using the SPARQL 1.1 SERVICE-clause to allocate a certain Basic Graph Pattern (BGP), called *service pattern*, to specified endpoints. Our approximation engine currently does not yet support UNION-clauses, OPTIONAL-clauses and filters outside of service patterns.

To approximate a join between two service patterns, a data synopsis of the data matching the first service pattern is generated by the responsible endpoint. This synopsis summarizes the bindings of the joining variables for each solution of the assigned service pattern. The data synopsis is generated by inserting the string representation of the bindings of a solution into a Bloom Filter. If multiple variables are joining, the bindings are combined into one string using a special delimiter. The endpoint responsible for the second SERVICE-clause receives the data synopsis and does a membership check on the

string representation of the bindings of the joining variables of its assigned service pattern. The bindings for which the membership check is positive form the basis for the *join synopsis*. The join synopsis summarizes the bindings of those variables which are joining with the next service pattern and is used as input for the next join approximation step.

To illustrate the approximation process, Figure 2.1 shows how the query in Listing 2.1 would be approximated. First, `ep1.com` receives the first service pattern, consisting of only one triple pattern `?a ex:p ?x`, and creates a list of bindings for variable `?a` (① in Figure). These bindings get approximated by an appropriate data synopsis ②. The synopsis is joined with the bindings provided by endpoint `ep2.com` for the second service pattern `?a ex:p ?b` ③. Note that only variable `?a` is involved in the join while a synopsis for the corresponding bindings for variable `?b` is created ④. The second synopsis is joined with the bindings for the third service pattern `?y ex:p ?b` ⑤. Since this clause is the last one, there is no further synopsis needed. Instead, we count the number of bindings that join with this last synopsis ⑥. This number is the estimated cardinality of the join between the three service patterns when they are assigned to the sources according to the federated query in Listing 2.1.

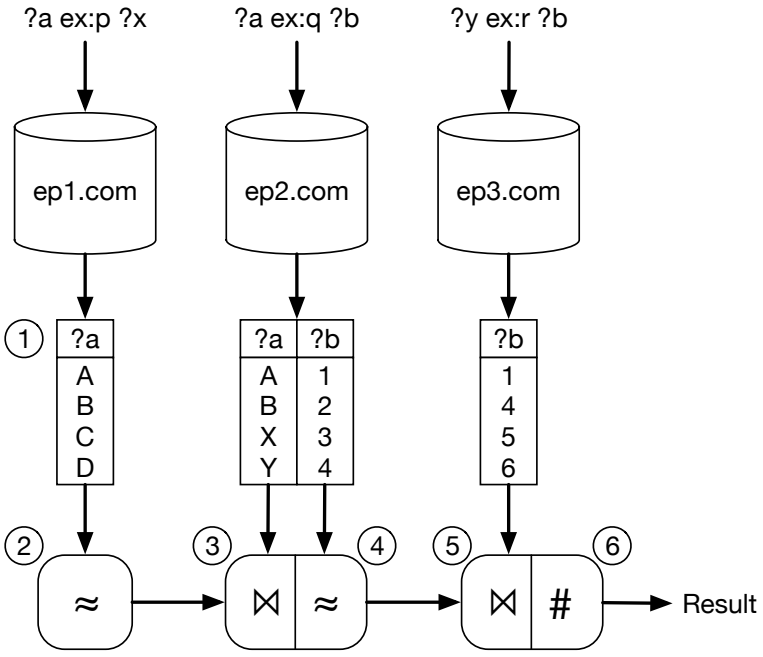


Fig. 2.1: Approximating the query in Listing 2.1 using our approximation engine.

Listing 2.1: A SPARQL query with 3 Service Patterns, each consisting of 1 Triple Pattern.

```
PREFIX ex: <http://example.com/>
SELECT * WHERE {
  SERVICE <http://ep1.com> {
    ?a ex:p ?x . }
  SERVICE <http://ep2.com> {
    ?a ex:q ?b . }
  SERVICE <http://ep3.com> {
    ?y ex:r ?b . }
}
```

2.3.2 Experimental Setup

For our evaluation, we investigated the scenario where each triple pattern must be sent to a different source. This means that it is not possible to form exclusive groups to speed up query processing/approximation, as proposed by [Schwarte et al., 2011a].

For the evaluation, we used FedBench [Schmidt et al., 2011] as a benchmark. FedBench consists of 25 queries and more than 200 million triples distributed over 9 datasets. Since we do not support UNION or OPTIONAL-clauses at the moment, we removed queries containing those clauses from our evaluations. To give a baseline for the execution time of the different approximation techniques, we executed each query using the query engine Jena ARQ[1]. We used the SERVICE-clause to direct each triple pattern to a separate SPARQL endpoint. We used Blazegraph[2] as a triple store. Table 2.1 shows the queries used, their runtime in milliseconds when using Jena ARQ, the actual cardinality of the query answer, and the number of triple patterns in the query.

We simulated both, query execution using Jena ARQ and the approximations using Bloom Filters, with a network speed of 10 Mbps, which is around the average speed of the top 10 countries in the world [Belson, 2015]. For the query execution, we adapted Jena ARQ to do block-nested-loop joins with a block size of 500 bindings to reduce the number of HTTP-connections, which have a negative impact on the runtime behavior of federated query execution. In addition, we optimized the join order of the different queries using simple heuristics to keep query execution in a reasonable time-frame. The query execution and all query approximations used the same join ordering to keep the results comparable.

For the Bloom Filter implementation, we used the Guava Google Core Library for Java[3].

2.3.3 Results

Figure 2.2 shows the absolute value of the *relative* error and the *relative* execution time of the approximation both computed with respect to the actual runtime and count when running the query in a federated fashion using Jena ARQ.

[1] https://jena.apache.org
[2] https://www.blazegraph.com
[3] https://github.com/google/guava

Table 2.1: The execution time, count, and number of triple patterns of the different queries.

Query	Time [ms]	Cardinality	Triple patterns
CD3	2.50E+03	2	5
CD4	3.90E+02	1	5
CD5	4.80E+02	2	4
CD6	1.30E+03	11	4
CD7	5.70E+02	1	4
LS3	3.80E+04	9054	5
LS4	5.20E+02	3	7
LS5	1.02E+02	393	6
LS6	4.40E+05	28	5
LD1	6.90E+02	308	3
LD2	4.70E+02	185	3
LD3	7.60E+02	159	4
LD4	3.10E+03	50	5
LD5	3.00E+02	28	3
LD6	6.20E+02	39	5
LD7	1.50E+03	1216	2
LD8	5.90E+02	22	5
LD9	3.20E+02	1	3
LD10	2.90E+02	3	3
LD11	2.00E+03	376	5

The relative error e_{rel} is defined as

$$e_{rel} = \frac{card_{est} - card_{actual}}{card_{actual}} ,$$

where $card_{est}$ is the estimated cardinality of the query answer based on the approximation and $card_{actual}$ is the actual cardinality of the query answer.

The relative execution time t_{rel} is defined as

$$t_{rel} = \frac{t_{est}}{t_{actual}} ,$$

where t_{est} is the runtime of the approximation technique and t_{actual} is the runtime of the query execution using Jena ARQ.

Clearly, a relative runtime of less than 1 is desirable, as otherwise it would be faster to execute the query and get the actual cardinality. For the relative error, it is not so clear what kind of error would still be in an acceptable range.

Each plot in Figure 2.2 shows the relative error (solid line) and relative execution time (dashed line). We measured the error and execution time for false positive rates of $fpp = 0.1, 0.01, 10^{-4}, 10^{-8}$.

As we can see in Figure 2.2, the runtime of the Bloom Filter approximation is very often disappointing. The approximation tends to require considerably more time for the approximation than the actual query execution. Surprisingly, the execution time for those approximations often improves when increasing the size of the underlying data synopsis. The discussions in Section 2.4 provide a good explanation for this behavior: the more accurate the synopsis, the less false positives must be processed. The overhead in processing more false positives seem to have a bigger negative impact on the runtime than the reduction of the size of the synopsis. *This behavior somewhat counteracts the actual purpose of a data synopsis to provide a trade-off between less accurate information and reduced processing time.*

Discussion of selected queries: The Bloom Filter approximation shows good results for the runtime of queries LS3, LS5, LS6, LD2, and LD4. Also, the error is comparably low and most of the time below 1. For those queries, the approximation can be considered successful: the approximation is able to return a reasonable approximation of the result size while running considerably faster than the actual query execution.

Queries CD7, LS4, and LD11 show worse approximation for a false positive probability of 10^{-8} than for a probability of 10^{-4}. One likely explanation for this is the fact that the original false positive analysis done by [Bloom, 1970] is incomplete and only gives a lower bound on the false positive rate. Indeed, as [Bose et al., 2008] points out, the actual false positive rate might be worse than expected when a small value for the false-positive probability is chosen as a parameter and a large number of hash functions have to be used in the filters.

The approximation yields a relative error of 0 for the query LD9. The reason for this behavior is that the last triple pattern only matches one single triple. Thus, our approximation engine predicts a cardinality of at most 1, because the prediction is based

on the number of those triples matching the last triple pattern which also join the synopsis of the previous joins, which can never be larger than the number of triples matching the last triple pattern. At the same time, the actual result of the query is also 1. Hence, approximation technique which overestimate the cardinality will yield a perfect prediction, necessarily. However, the relative runtime of the approximation methods is around 1.

The query LD4 is another one where the last triple pattern only matches one single triple. Again, our approximation engine predicts a cardinality of at most 1. But this time, the actual result is not 1 but 50. In fact, all 50 different results have the same binding for the last joining variable. As the Bloom Filter does not account for duplicated values the approximation wrongly predicts 1 instead of 50. At the same time, the approximation speed profits slightly from this error by yielding a faster execution time.

2.4 Theoretical Analysis of the Cumulative Join Estimation Error

In this section, we investigate to theoretical foundations which can explain the disapointing performance of our Bloom Filter join approximation. We will estimate the cumulative error for WoD queries for approximation techniques that overestimate the results due to false positives, which includes all data synopses which are not based on sampling, in particular, our Bloom Filter-based method. Such overestimating data synopses can lead to false-positive matches (i.e., the prediction of a match where there is none) due to loss of information. When approximating multiple joins, the result of the first join (including its false positives) is again encoded as a data synopsis passed to the second join, which will now attempt to match all encoded elements including the false-positives. Hence, the error of the synopsis gets propagated through each join and accumulates [Ioannidis and Christodoulakis, 1991].

We now formally discuss the propagation of the error in a multi-join, that is, a sequence of joins where the output of one join is an input for the next join. For this we extend the formula for the error derived by [Ioannidis and Christodoulakis, 1991] by analyzing the relation between the rate of false positive matches and the join selectivity based on the following assumption:

Assumption 1. *All joins are equality inner-joins.*

Assumption 1 is motivated by the fact that we do not consider filter expressions in our evaluation and hence, we only support equality joins. We will not discuss outer joins because their cardinality estimation is trivial.

Assume we want to approximate the join result of joining $m+1$ basic graph patterns $bgp_0, \ldots, bgp_m$. We define n_i for $i \in \{0, \ldots, m\}$ as the number of results selected by BGP bgp_i from the corresponding dataset. Let n_i^{FP} be the number of false positives at step i, which is the number of elements that are wrongly classified as a match given the synopsis from the previous joins. We define the false positive rate fpr_i as the ratio between n_i^{FP} and n_i.

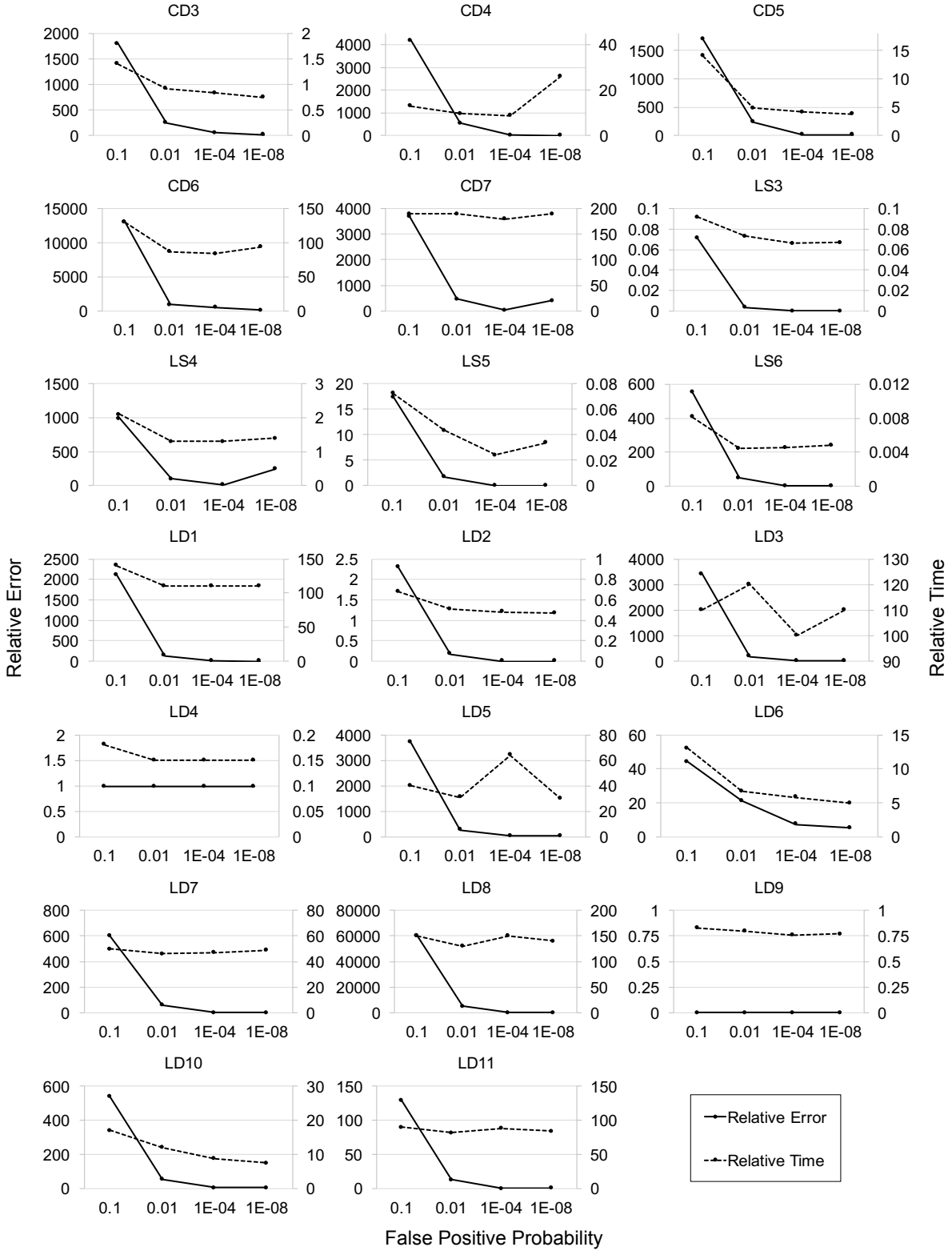


Fig. 2.2: Relative error (left vertical axis) and relative execution time (right vertical axis) for different false positive probabilities.

Let $prop_i^{FP}$ for $i \in \{1, \ldots, m\}$ be the *propagation rate* of the false positives in the synopsis for the join approximation between $bgp_0, \ldots, bgp_{i-1}$. The propagation rate indicates how many false-positives matches are produced *on average* by a single false-positive propagated from previous join approximations.

The expected number of false positives FP_k for the approximation of the join of $bgp_0, \ldots, bgp_k$ is the number of false positives introduced by fpr_k for bgp_k plus the false-positives given the false-positives FP_{k-1} of the approximation of the join of $bgp_0, \ldots, bgp_{k-1}$:

$$FP_k = \underbrace{fpr_k \cdot n_k}_{\text{synopsis error}} + \underbrace{prop_k^{FP} \cdot FP_{k-1}}_{\text{propagated error}} \quad . \tag{1}$$

We define $FP_0 := 0$, as there is no propagated error influencing the first join operation. Applying Equation 1 recursively gives the following formula for the number of false positives FP_m of the approximation of the join of $bgp_0, \ldots, bgp_m$:

$$FP_m = \sum_{i=1}^{m} fpr_i \cdot n_i \cdot \prod_{j=i+1}^{m} prop_j^{FP} \quad . \tag{2}$$

To compare the number of false positive matches with the number of true positive matches, we analogously compute the number of true positives TP_m. To do that we define the propagation rate of the true-positives $prop_i^{TP}$ for the join between $bgp_0, \ldots, bgp_{i-1}$ (just like we defined the propagation rate $prop_i^{FP}$ for false-positives). Using this definition, the number of true positives TP_k of joining $bgp_0, \ldots, bgp_k$ is:

$$TP_k = n_0 \cdot \prod_{j=1}^{k} prop_j^{TP} \quad . \tag{3}$$

To continue our analysis, we introduce the following assumption:

Assumption 2. *False-positive matches and true-positive matches have the same propagation rate, i.e.* $prop_j^{FP} = prop_j^{TP} =: prop_j$.

Assumption 2 is motivated by the fact that the propagation rate of both, true positives and false positives, are influenced by the type of URI and not by the fact whether they are false or true positive. For example, there is an average number of addresses joining with a person, independent of whether the person is true or false positive. Hence, we assume that there is no bias which would cause that a true positive match has, on average, a lower/higher propagation rate than a false positive match.

Under Assumption 2, we get the following formula for the relative error E of the approximation:

$$E = \frac{FP_m}{TP_m} = \frac{\sum_{i=1}^{m} fpr_i \cdot n_i \cdot \prod_{j=i+1}^{m} prop_j}{n_0 \cdot \prod_{j=1}^{m} prop_j} = \sum_{i=1}^{m} \frac{fpr_i \cdot n_i}{n_0 \cdot \prod_{j=1}^{i} prop_j} \quad . \tag{4}$$

We define the selectivity $sel_{bgp_0,\ldots,bgp_j}$ of the join between $bgp_0, \ldots, bgp_j$ as the number of results of the join divided by the product $n_0 \cdot \cdots \cdot n_j$, i.e. the cardinality of the cross product of all results for $bgp_0, \ldots, bgp_j$. It follows that:

$$n_0 \cdot \prod_{j=1}^{i} prop_j = sel_{bgp_0,\ldots,bgp_j} \cdot \prod_{j=0}^{i} n_j \tag{5}$$

and consequently:

$$E = \sum_{i=1}^{m} \frac{fpr_i}{sel_{bgp_0,\ldots,bgp_i} \cdot \prod_{j=0}^{i-1} n_j} \quad . \tag{6}$$

Equation 6 shows that the higher the number of joins and the lower the join-selectivities $sel_{bgp_0,\ldots,bgp_j}$ are, the smaller the false positive rate fpr_i of the approximation must be to produce a reasonably small estimation error. Thus, the *approximation error is determined by the* ***ratio between the false positive rate and selectivity*** and not "just" the false positive rate. In addition, this error does not only lead to inaccurate results but *also has a negative impact on the execution time* of the approximation: If the selectivities are low and the false-positive rates relatively high, it can happen that the query approximation mainly processes false-positives and that the data synopses based on these false-positives are larger than the actual data of all true-positives. Thus, the *query approximation might take longer than the actual query execution.*

Note that these theoretical findings should be cause for concern for building federated query systems in the light of false positive baring data synopses. In the next section, we will explore if these theoretical considerations apply to the practical Web of Data setting that we are currently exploring in federated querying.

Verification of the Analysis: We want to verify that our theoretical analysis indeed serves as an explanation of the error and runtime behavior that we observed in Section 2.3. For this, we compared the estimated error predicted by our analysis with the actual error which we observed in our evaluation. Figure 2.3 plots the estimated error based on equation 6 against the actual relative error measured for the Bloom Filter approximation in a log-log scale (as the values include both very small and very large numbers). Figure 2.3 suggests a very strong correlation between relative error of the estimation and the

predicted error by our analysis. Indeed, both the Pearson correlation coefficient $R^2 = 0.81$ and the Spearman's Rank Correlation $\rho = 0.76$ between the actual (non-log) numbers indicate a strong correlation between the theoretical estimation of the error and the actual evaluation. Not included in the figure, but included in the calculation of the correlation coefficients are those estimates that produced a relative error of 0, which could not be drawn in the log-log scale plot.

The figure shows that for a false positive probability of 10^{-8} (indicated by little pluses "+" mostly at the top left of the Figure) the actual error is not as small as one might expect. One likely explanation for this is the fact that the specified false positive rate only gives a lower bound on the actual false positive rate, as we already discussed in Section 2.3.

Overall, Figure 2.3 confirms the theoretical analysis of the error accumulation in Equation 6, which indicates that *SPARQL queries require data synopses with very low false-positive rates to produce reasonably accurate results –– an effect which is much more pronounced for queries with a low selectivity, as we have shown in Equation 6.* This, in turn, might require specific implementations of Bloom Filters that can handle such low probabilities. However, the need for such accurate synopses make it questionable whether join approximations that produce false positives are suitable for such tasks, in general.

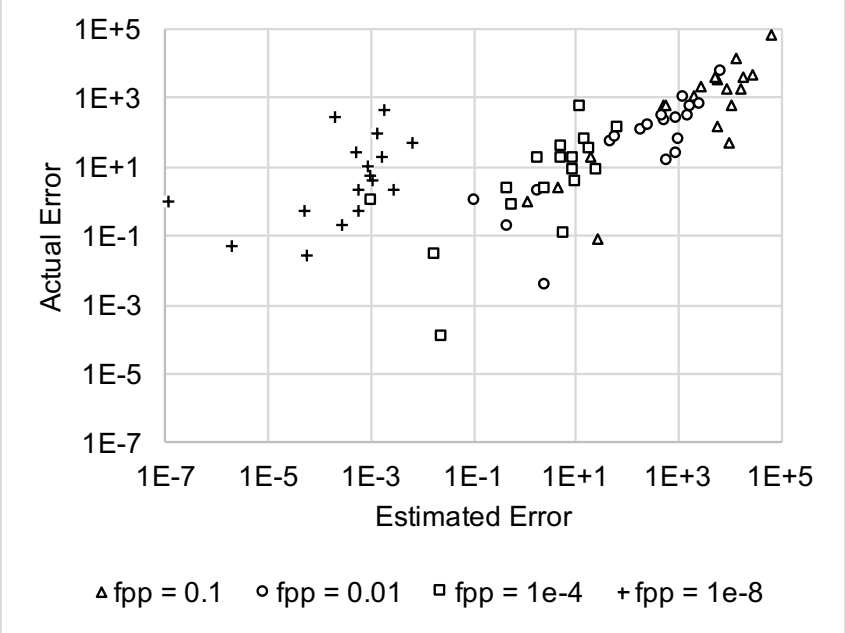


Fig. 2.3: Estimated error plotted against actual error.

2.5 Limitations and Future Work

In this paper, we focused our evaluations on queries which did not include UNION, OPTIONALs, and FILTERs outside of service patterns. However, we think, given the perfor-

mance of our Bloom Filter approximation in this simpler setting, one cannot expect the approximations to perform better when extending the evaluation to include more complex queries. In particular, joins over multiple variables are likely to further constrain a join offering even more potential for synopses to generate false positives. UNIONS can be seen as a conjunction of multiple queries, which does not pose a significantly different setting. We will have to consider OPTIONALs in future work, though our intuition indicates that they can be seen as a combination of two different queries, which should not impact our results. FILTER expressions are more complex and warrant future work, as they might impact synopses construction. In particular, when a filter compares two bindings from different sources it can only be applied after the join, which may require executing it on the actual data (rather than only on a synopsis), anyway.

Obviously, Bloom Filters represent only a one possible method to estimate the join-cardinality of SPARQL queries. Our theoretical considerations, however, are based on the fact that most synopses have false positives, so we do expect these findings to generalize.

Our assumption that each triple pattern must be sent to a different source results in a high number of joins between different endpoints. In practice, it could be that many queries may not have to be distributed to such an extent and subqueries with multiple triple patterns may be answered by a single endpoint. As the WoD grows, however, we are likely to see a rising number of queries that are getting bigger and are increasingly distributed. Hence, we believe that our findings do point to a core problem of federated querying on the Web of Data.

2.6 Conclusion

This paper set out to investigate the applicability of query approximation for source selection. We hypothesized that the performance of cardinality approximations of federated SPARQL queries degenerates when applied to queries with multiple joins of low selectivity. Indeed, both our empirical evaluation and our theoretical considerations indicate that data synopses are not suitable for this task due to their cumulative error, which also substantially slows down the estimation process. Based on our analysis, one can only expect good approximation performance if (1) the number of joins is low, (2) the join-selectivity is high, and/or (3) there is a bias which causes true positive matches to have a much higher propagation rate than false positive matches. These findings seriously hamper the usefulness of current selectivity estimation techniques for domains such as the WoD, where the number of joins involved in the estimation process is high. Indeed, our focus on a setting with many joins pinpointed a deficit in the generalizability of selectivity estimation techniques which came from a domain where usually only few inter-domain-joins are to be expected.

It is important to note that whilst this paper focused on federated SPARQL-querying in the context of the WoD our findings generalize to any federated conjunctive querying setting where join estimates cannot be precomputed.

The consequence of our work is twofold: First, to fulfil the Semantic Web vision via federated querying requires a *renewed effort to find suitable join-approximations for federated*

SPARQL queries. As the WoD progresses, we will require more sophisticated approximation techniques, which are more adapted to the WoD: i.e., the need to be able to handle many inter-source joins and low selectivity better. Note, however, that no matter what new technique gets introduced, in the presence of low selectivity, our analysis of the error propagation adds a limit to what can be achieved by join-approximations that cause false-positives.

Second, if the community does not manage to drastically improve approximation techniques, *there might be a need to consolidate datasets from different sources into more centralized structures to reduce the number of endpoints that must be accessed during federated query execution*. This centralization will allow computing better estimations or incorporating them into the indices.

In conclusion, our findings showed that we may have to rethink well-known techniques such as the concept of join-approximation when applying them to the WoD. Doing so, will both advance our understanding of these techniques and may cause us to rethink the structure of the Web of Data as a whole.

Chapter 3

Selling Data in a Federated Fashion

This chapter is based on:

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. 2018. ***FedMark: A Marketplace for Federated Data on the Web****. arXiv:1808.06298v1 [cs.DB]*

FedMark: A Marketplace for Federated Data on the Web

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken

Department of Informatics, University of Zurich, Switzerland

Abstract. The Web of Data (WoD) has experienced a phenomenal growth in the past. This growth is mainly fueled by tireless volunteers, government subsidies, and open data legislations. The majority of commercial data has not made the transition to the WoD, yet. The problem is that it is not clear how publishers of commercial data can monetize their data in this new setting. Advertisement, which is one of the main financial engines of the World Wide Web, cannot be applied to the Web of Data as such unwanted data can easily be filtered out, automatically. This raises the question how the WoD can (i) maintain its grow when subsidies disappear and (ii) give commercial data providers financial incentives to share their wealth of data. In this paper, we propose a marketplace for the WoD as a solution for this data monetization problem. Our approach allows a customer to transparently buy data from a combination of different providers. To that end, we introduce two different approaches for deciding which data elements to buy and compare their performance. We also introduce FedMark, a prototypical implementation of our marketplace that represents a first step towards an economically viable WoD beyond subsidies.

3.1 Introduction

Inspired by the WWW's characteristics, the *Web of Data (WoD)* is a decentralized repository of data, where many data tenants publish and manage interlinked datasets whether indexed or not. Its goal is to provide a globally distributed knowledge base where query engines can mix-and-match data from various, distributed data-sources towards answering queries. Most current datasets on the WoD are freely available, either subsidized by governments via data access laws and research grants or maintained by enthusiasts. Some provider of datasets will be able to maintain funding for their datasets and continue to be open. Right now, however, only a third of the public SPARQL endpoints have an uptime of 99% and above [Buil-Aranda et al., 2013b]. Without financial incentives, many promising datasets will be poorly maintained or unavailable as there is no one willing to invest time and money to keep the data up-to-date and the endpoint running [Van Alstyne et al., 1995].

Unfortunately, most incentive mechanisms from the WWW do not translate to the WoD, as data is queried by machines rather than humans. Consequently, WoD query results often do not contain any attribution to the original source and algorithms can simply filter out any contained advertisement removing most non-monetary benefits from the publisher. Hence, the many motivations typically entailed in authoring a web page—fame, money through advertisement, acknowledgment, or recognition—do not carry over to the WoD. Even though provenance techniques exist, such meta-information will not be shown to the user if not explicitly requested.

Example 1. Consider an Intelligent Personal Assistant (IPA)—a computer program assisting a user by automatically searching the WoD for relevant information and interacting

with other computer programs—which searches the WoD on a daily basis for relevant information regarding error messages that occur while working with computers (akin to paring osquery[1] with an error recognition and suggestions database extracted from stackoverflow). The IPA uses context information about the program throwing the error, the operating system, and other relevant data to find articles, comments, and other pieces of information that can help understanding and solving the problem which caused the error. Unlike a keyword-based search in the WWW, the IPA could find information that is much more specific to the context of the error and provide different suggestions without any interaction needed by the user.

IPAs like the one presented above are one of the big promises of the Web of Data [Berners-Lee et al., 2001]. However, the question we raise in this paper is how people can be motivated to create the content needed to enable such a vision. In our example, many of the articles, comments, etc. about problems are written by fellow users, who already encountered the problem and are now sharing the gathered knowledge. In the WWW, such fellow users are credited when giving a helpful answer and are usually thanked by others. The website hosting the platform for this knowledge exchange makes money by showing advertisement and job offers to the users. As we can see, there are two *incentive mechanisms* which keep the knowledge exchange platform alive: (1) *acknowledgement* and *reputation* incentivize users to share their knowledge and (2) *money* from advertisements and job offers help to finance the platform.

In our IPA scenario, however, the users creating the knowledge and the platform offering the knowledge are transparent to the end-user who is consuming the knowledge through the IPA. In fact, the lack of any end-user interaction required is one of the big advantages of having an IPA in the first place. But how can such a scenario work if we remove the incentive mechanisms which were used in the WWW to create the necessary knowledge?

Due to the above-mentioned reasons, we believe that in the long-term many providers of semantic data have to charge fees (directly or indirectly) for accessing their data in order to finance their services. However, as soon as users are charged for the data they consume, economic considerations play an important role during query planning. In particular, the users (or programs acting on behalf of the users) have to decide to which data sources they buy the right for accessing. This decision, in turn, depends on how much the bought data can contribute to a specific query. As we showed in [Grubenmann et al., 2017a], it is very difficult to decide *before* query execution how much a certain source can contribute to a query answer. However, *after* query execution it is too late to decide against the inclusion of some sources, as the data is already bought. Whilst data synopses can sometimes help in deciding which data sources might be worth accessing for a specific query, our analysis showed that there is no universal approximation method which can consistently yield good enough results to judge the economic utility of a source for a specific query. *These findings question the practicability of a scenario, where data providers charge customers directly for accessing their data.*

[1] https://osquery.io

Alternatively, one might argue, the nature of data will lead to natural monopolies and we should concentrate on building *one large centralized database*. Such a database would allow the maintainer to extract monopolistic fees for its usage, which could pay for the data maintenance. For example, data services such as *Bloomberg*, *LexisNexis*, or *Thomson Reuters* charge customers high fees for accessing their data primarily using a subscription-based model. These sellers can price their services by calculating a quasi-monopolistic price on their whole data offering [Bakos and Brynjolfsson, 1999]. Indeed, most non-monopolistic settings struggle to find a good pricing-scheme. The *Azure DataMarketplace* [Microsoft Corporation, 2011], e.g., closed in March 2017, due to the lack of attraction. The Copenhagen City Data Exchange is still trying to figure out how to find a good way to price their data sets.[2] However, none of these solutions provide their data in a way such that they can be queried in a federated fashion. They do not provide the means to join datasets from multiple providers and access can only be purchased in an all or nothing approach, thereby forgoing the complementarities the WoD would enable. This is a serious drawback, because customers are often interested in a specific *combination* of data from different providers that are joined in a certain way. Also, as Van Alstyne et al. [Van Alstyne et al., 1995] argue, the incentive misalignments in a federated system based on these principles may lead to significant data quality problems. Finally, some users may not be prepared to pay for the large bundles of data sold by these monopolists as they are only interested in occasional or very partial access. These are left out of these markets. Hence, the central question of this paper is *how can we facilitate a financial sustainable and decentralized WoD without government subsidies or federation-averse centralization and fulfill the promise of the data economy [Bublies, 2017]?*

This paper proposes FedMark, **a marketplace for data following the WoD principles of federated querying**. In contrast to the settings described above, FedMark allows a user (or customer) to submit a query and decide *after* query execution which data should be bought without accessing the data (and incurring the moral hazard of not wanting to pay for already seen data). To this goal, FedMark acts as a mediator between the customer and various data providers. FedMark executes the query but does not pass the query answer to the customer, yet. Based on a *summary* of the full query answer, the customer can decide which parts of the query answer to buy. This selection of a subset of the query answer—which we call an *allocation*—can be done either manually by composing the query answer based on personal preference or automatically by using an *allocation rule*, which automatically determines how the query answer should be composed based on the available information. As manual composition is impractical in most large settings, allocation rules are crucial tools to deal with the exponentially growing number of possible allocations.

FedMark introduces a new paradigm towards querying and pricing the WoD relying on a market-based approach and principles of federated querying. This paradigm enables data providers to finance their wealth of data without relying on subsidies but on per-query fees. Our approach has the following advantages:

[2] https://www.citydataexchange.com/, personal communication

- A customer can buy a query answer from a *combination* of different data providers in a transparent way.
- Given a customer's query, our marketplace creates a query answer based on all available datasets from which a customer can allocate his most preferred subset.
- The price for the allocation depends only on the data *contributing* to the allocation. Especially, the price is independent of query execution (in particular, the order of joins) and size of underlying datasets. Hence, FedMark compensates data providers for the value of the data they contribute for the specific query answer.

Our contributions are, hence:

- the introduction of a market-based paradigm towards querying and pricing,
- the presentation of two different allocation rules for such a marketplace,
- the introduction of a prototype system FedMark implementing this paradigm, and
- the thorough evaluation establishing the practicality of our approach in terms of runtime overhead and utility maximization.

In the following, we start with some preliminaries about the Web of Data. We continue with the discussion of related work and then introduce our data market concept. This leads the way to our prototype implementation FedMark and the introduction of two allocation rules. Next, we perform an empirical evaluation of the runtime of the introduced allocation rules. We close with a discussion of the limitations of this study and an outlook for future work.

3.2 Preliminaries

In the Web of Data, relations between resources are modelled using the Resource Description Framework (RDF) [Cyganiak et al., 2014], using *resources* and *literals*. A resource can denote anything, e.g., a website, an image, a physical thing, or an abstract entity. A literal is a string of characters with an optional datatype associated to it. The relations between resources and literals are modelled as statements consisting of a subject, object, and predicate linking the former two. A statement in RDF is also called a *triple*. Fig. 3.1 shows an example of how different information about a hotel can be modelled using such triples. Subjects and objects are illustrated with circles, predicates with arrows pointing from subjects to objects. Labels with quotation marks indicate literals, other labels indicate resource identifiers.

Users interested in the data represented in an RDF graph can use the query language SPARQL [Harris and Seaborne, 2013]. Listing 3.1 shows an example of how a user could ask for images of a hotel named "Hotel California". Identifiers starting with a "?" indicate variables. Each line ending with a "." indicates a *triple pattern*, which can be matched to triples inside a graph. Such triple patterns can be used to form more complex *graph patterns*, which can be combined with other operators like filter expressions, joins, and unions. The answer to a SPARQL query, which we refer to as *query answer*, consists of bindings to the variables of the graph pattern after the WHERE clause projected to the variables specified after the SELECT clause. A *SPARQL endpoint* is a web address which

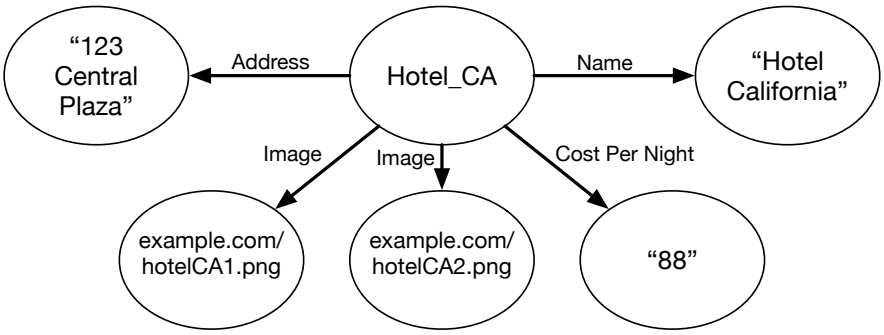


Fig. 3.1: An example RDF graph describing a hotel.

accepts SPARQL queries and returns query answers based on the RDF graph stored in the back-end.

Table 3.1 shows the bindings which would be returned as query answer if the SPARQL query from Listing 3.1 would be executed against the RDF graph from Fig. 3.1. Each row in the table represents a single *solution mapping* to the query. A solution mapping is one possible set of bindings of variables to resources or literals which correspond to the queried RDF graph. A query answer is a set of such solution mappings. A query answer can contain one, multiple, or no solution mappings, depending on the RDF graph against which the query was executed [Harris and Seaborne, 2013].

Listing 3.1: A query which asks for images for a hotel named “Hotel California”.

```
PREFIX ex: <http://example.com/>
SELECT ?image
WHERE {
        ?hotel ex:depicts ?image .
        ?hotel ex:name "Hotel␣California" . }
```

Table 3.1: Result of the query in Listing 3.1.

?image
example.com/hotelCA1.png
example.com/hotelCA2.png

It is possible to execute a query against multiple RDF graphs. Different RDF graphs can be made available on a single SPARQL endpoint or on different endpoints. In the latter case, a SPARQL query must be split up into subqueries which must be executed on the different servers. In this case, the different endpoint can be combined into a *federation*

of SPARQL endpoints. The (partial) query answers returned from the different machines must be joined together to form the final query answer.

3.3 Related Work

Our approach is based on standardized federated querying on the WoD [Buil-Aranda et al., 2013a]. As such, it relies on basic techniques of SPARQL querying [Pérez et al., 2009]. Here, we very succinctly discuss the most recent federated querying techniques before elaborating on previous attempts of pairing market-based ideas in data management.

Federated Querying on the WoD: The traditional concepts for federated RDF querying provided integrated access to distributed RDF sources controlled by the query engine [Erling and Mikhailov, 2009, Harth et al., 2007, Quilitz and Leser, 2008]. The drawback of these solutions is that they assume total control over the data distributions—an unrealistic assumption in the Web. Addressing this drawback, systems were proposed that do not assume fine-grained control: some exploit perfect knowledge about the `rdf:type` predicate distribution [Langegger et al., 2008] while others proposed to extend SPARQL with explicit instructions controlling where to execute sub-queries [Zemánek et al., 2007]. Often, however, the query writer has no ex-ante knowledge of the data distribution.

SPLENDID [Görlitz and Staab, 2011] proposed to exploit service descriptions and VoID statistics about each endpoint, to perform source selection and query optimization. HiBISCuS [Saleem and Ngonga Ngomo, 2014], on the other hand, maintains an index of authorities for certain URIs. FedX [Schwarte et al., 2011b] uses no knowledge about mappings or statistics about concepts/predicates. It consults all endpoints to determine if a predicate can be answered (caching this information for the future). Fed-DSATUR [Vidal et al., 2016] is an algorithm for SPARQL query decomposition in federated settings without relying on statistics, indices, or estimates for source selection. Forgoing any ex-ante knowledge about data sources and any requirements on data storage, Avalanche [Basca and Bernstein, 2014] proposes an approach that combines data-source exploration followed by extensive parallelized and interleaved planning and execution.

Following another avenue, Hartig et al. [Hartig et al., 2009] describe an approach for executing SPARQL queries over Linked Open Data (LoD) based on graph search. LoD rules, however, require them to place the data on the URI-referenced servers—a limiting assumption, e.g., when caching/copying data.

Whilst these approaches provide a solid foundation for federated WoD querying, none of them considers the economic viability of their proposed solutions. Hence, we will extend this foundation with a market-based allocation approach to ensure economic viability.

Market-based Approaches towards Resource Allocation in Computational Systems: The idea to use markets to allocate computational resources is almost as old as computers. Already in the 1960s, researchers used an auction-like method to determine who gets access to a PDP-1, the world's first interactive, commercial computer [Sutherland, 1968]. Since then, many market-based approaches for computational systems have been proposed.

Early research on market-based scheduling focused on the efficiency of computational resource allocation. The Enterprise system [Malone et al., 1983] introduced a market for

computational tasks. It efficiently allocated the tasks to multiple LAN-connected nodes, where task processors broadcast requests for bids and bid on tasks. Likewise, Spawn [Waldspurger et al., 1992] utilized a market mechanism to optimize the use of idle resources in a network of workstations. More recently, [Lai et al., 2005] proposed Tycoon, a distributed computation cluster, featuring a resource allocation model. The authors claim that an economic mechanism is vital for large scale resource allocation—a common problem on the Web. Furthermore, [Auyoung et al., 2006] demonstrates how profit-aware algorithms outperform non-profit aware schedulers across a broad range of scenarios.

In data processing centric scenarios, [Labrinidis et al., 2007] applied market-based optimizations to real-time query answering systems. [Stonebraker et al., 1996] proposed a WAN-scale Relational Database Management System with a market-based optimizer instead of a traditional cost-based one. [Dash et al., 2009] proposed a market-based approach for cloud cache optimization taking into account a user's value for getting an answer to a query. However, their approach focuses on the cost-side of cloud computing.

For relational databases, markets for SQL queries were proposed which sell data instead of computational resources for answering queries and use arbitrage-free pricing schemes to calculate payments [Deep and Koutris, 2017, Koutris et al., 2013]. [Wang et al., 2016] proposed an auction mechanism for data which considers the negative externalities of allocating data to different buyers. However, they do not consider the possibility of joining datasets from different providers, which is an important aspect of the scenario we are investigating. To the best of our knowledge, none of these systems considers partial answers due to budget constraints or the possibly differing valuations of various users/-queries, which is very typical in the WoD.

As a precursor to our research, we conducted a pilot study simulating a market platform for the WoD [Zollinger et al., 2013]. This paper here represents a significant rework of the old pilot as it proposes a complete model, an improved market analysis, and a prototype implementation instead of a simulation. In [Moor et al., 2015], we introduced the idea of using a double-auction for the WoD and showed the deficiency of the threshold rule in this setting together with three ways to correct them. However, our approach assumed that we have access to accurate join-estimates to produce satisfying results – an assumption which might be hard to enforce in the WoD. In [Grubenmann et al., 2017b], we presented our vision of a marketplace which allows customers to buy data from decentralized sellers in an integrated way. In this paper, we fulfil this vision and present our implementation of a federated marketplace for such decentralized data.

3.4 Market Concept

We begin to describe our market concept by continuing Example 1 from Section 3.1. Throughout this paper, we will extend this example to show how to decide which solution mappings to include in the customer's query answer and how much the customer must pay for it.

Example 2. Consider a user who encounters the error message `"0x12345678"` while working with the (fictive) Integrated Development Environment (IDE) "Hack IDE" on a

Java program. The IPA will recognize that the problem occurred and search for suggestions to fix the problem related to the error message. The IPA will order the different suggestions to the problem by their success rate. The IPA will use other information available, like the operating system, to refine the search and to get only relevant suggestions. Listing 3.2 shows how a SPARQL query generated by the IPA might look light. Each row of the query answer represents one possible suggestion to the problem with the corresponding success rate.

We assume that at least part of the data needed to answer the query requires a payment from the user. In order to autonomously retrieve the query answer, the IPA buys the required data on behalf of the user in our marketplace. The marketplace finds the providers that offer datasets to answer this query. There might be multiple combinations of providers that would yield a non-empty query answer. Some of them might provide only suggestions without success ratings; others might provide only success ratings for suggestions, and some might provide both. As a result, there are multiple different combinations of datasets which produce (possibly) different query answers. Some of the query answers may contain only a few suggestions and ratings, whereas others may contain many, or none.

Listing 3.2: A SPARQL query asking an IPA can use to retrieve suggestions to a problem indicated by an error message.

```
PREFIX ex: <http://example.com/>
SELECT ?suggestion ?rate WHERE {
  ?suggestion ex:success_rate ?rate .
  ?suggestion ex:err_code "0x12345678" .
  ?suggestion ex:program ex:hack_ide .
  ?suggestion ex:language ex:java .
  ?suggestion ex:os ex:os_x .
} ORDER BY DESC(?rate)
```

At the core of FedMark lies the ability for a customer to join data from different providers to buy solution mappings to a given query. Instead of buying all the solution mappings contained in a query answer, FedMark allows a customer to select a subset of the solution mappings—which we call an *allocation*—and only paying the price of the allocated solution mappings.

Note that an allocation is also a query answer. We will refer to the result of the query execution as *query answer* and the result of the allocation process as *allocation* to emphasize the difference.

For a specific query, different combinations of providers' data might result in different (even empty) query answers. Our marketplace needs to (1) enable the customer to make an informed decision about which solution mappings to include into the allocation and (2) decide how much money has to be paid to each provider.

We use the following definition throughout the paper to denote single solution mappings, query answers, and allocations:

Definition 1 (Solution Mapping and Query Answer). *We denote as Ω the set of all possible solution mappings. We denote as $\omega \in \Omega$ a single solution mapping. A* query answer *$\rho \subseteq \Omega$ is a set of solution mappings.*

Definition 2 (Allocation). *An* allocation $a \subseteq \rho$ *is a set of solution mappings which are chosen from the query answer* ρ*.*

We now introduce the four different entities our market concept brings together, *providers*, *hosts*, *customers* and the *marketplace*, all depicted in Fig. 3.2. In the following, we further elaborate on these entities before introducing an operationalization of our marketplace in Section 3.6.

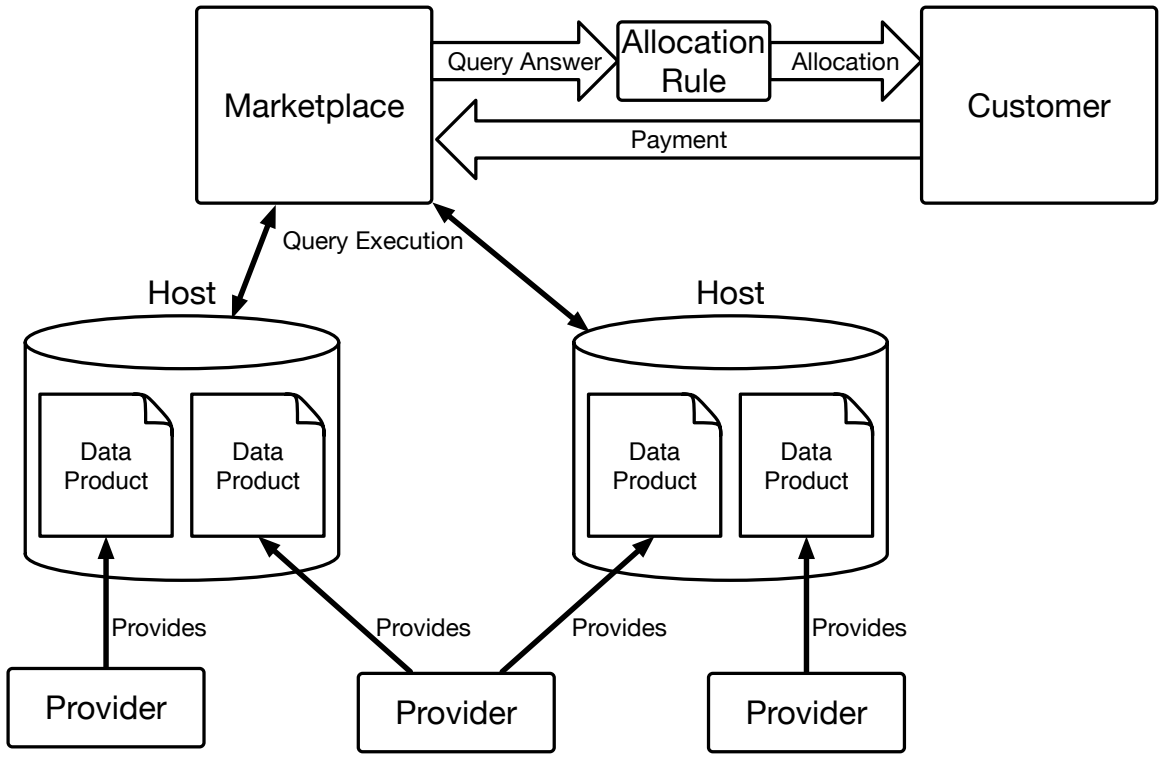


Fig. 3.2: Customer, Host, and Provider in the marketplace.

3.4.1 Provider

A provider is the originator of some data, which is used in the production of a query answer. Providers can group their data into different *data products* having different prices *per triple*. On the data level, a data product is just a collection of RDF-graphs, which are accessible under the same license. In addition to the price, data providers can specify other meta-data that might be relevant for customers such as terms/conditions of access, recentness, or origin of the underlying raw data. Providers are responsible for the *quality* of data, including *recentness*, *consistency* and *accuracy* [Dustdar et al., 2012].

Definition 3 (Data Products and Prices). *We will denote data products with* $P_1, \ldots, P_k$ *and their price with* $\pi_1, \ldots, \pi_k$*. Each Product* P_i *consists of a collection* G_i *of RDF graphs.*

The providers set the optimal price based on market conditions. This price can be learned, e.g., using reinforcement learning. The price π_1 of the data product P_i indicates the payment that is involved when using an *RDF-Triple* contained in the data product for an allocation. This is an important aspect of our market concept: A provider is only payed for those triples which are *allocated*. Triples which are accessed during query execution but not allocated, do not receive a payment from the customer. The reasons for FedMark to only charge for allocated triples are the following:

1. If customers would have to pay for all the accessed triples, allocated or not, they would "loose" some of the paid triples by operations like joins, projections, and filters. Hence, it could be more beneficial to omit filters and projections and perform (left) outer joins to get as much data as possible for the *same* price. This would incentivize customers to obtain data they do not need and optimize the query for the best data yield for a given price. We do not believe that it is desirable to create a market where such meta-optimizations are required from a customer to maximize the utility they can get out of our market.
2. If providers would be compensated for accessed triples, the same query could yield very different revenue for a seller depending on query execution. Especially in join-operations, constraints from previous joins can vastly reduce the data that has to be accessed. This means that providers whose data is accessed earlier during query execution have the tendency to sell more data. In extreme cases, the price of an allocation is mainly dominated by the provider who is accessed first which, in turn, also receives the majority of the revenue. We believe that this potential for very *unbalanced* vending of data is not desirable.
3. Compensating accessed triples would create wrong incentives for the data providers: they would be able to increase revenue by increasing the amount of accessed triples, even if those additional triples do not contribute in any way to the final query answer. Providers would basically be encouraged to produce as much "dead weight" data – data which is not really useful but has to be accessed during query execution because their lack of any use is only discovered after query execution – as possible to maximize revenue.
4. Charging for accessed triples adds an additional layer of complexity for query planning and optimization. In particular, the marketplace would have to estimate how many triples have to be accessed to calculate the price of a specific allocation prior to query execution. However, as we have shown in [Grubenmann et al., 2017a], such estimations can be very unreliable. A high discrepancy between estimated price and actual price can be very devastating for such a marketplace.

Note that providers do not serve their data; this is done by separate entities, the hosts. The separation between host and provider allows for more flexible business models for data provision, as some providers might have an initial budget to create some data

(e.g., government subsidies) but do not have the funds to cover the *operating costs* for running a SPARQL endpoint or may have other reasons to outsource the actual data provision. Providers can decide to act at the same time as a host for their own and/or other provider's data. Nevertheless, the market distinguishes between the two different roles, provider and host, and treats them as separate entities.

Data providers rely on the hosts to make their data available to the marketplace and thus, enable customers to buy their data. Similar to a Webhost for traditional Web content, the hosts in our market concept are paid by the provider based on some service agreement. Hence, the providers have to include the hosting costs into their pricing decision.

Similar to other digital goods such as software, eBooks, or digital music, the customer does not buy the good itself but buys the *right* to use it under certain terms. For example, most usage rights for digital goods do not allow their resale to third parties. It is, however, the task of the provider to specify the exact terms under which the specific good is sold.

We elaborate on the role of providers in the following example:

Example 3. Data providers who want to contribute data to the query introduced in Example 2 must offer data products which include data about suggestions to the programming problems or success ratings of those suggestions. Every query answer that requires data from one or several such data products results in some payments for the data providers.

Consider a data product P_A with a price of \$0.10 per triple providing success ratings for suggestions. This means that P_A can offer triples matching the first triple pattern in Listing 3.2. P_A is basically running a service where users are reporting on the failure or success of certain suggestions. Consider further two data products P_B and P_C with prices of \$0.02 and \$0.03 per triple, respectively, providing a database with the actual suggestions to various problems. P_B and P_C both can offer triples matching all but the first triple pattern in Listing 3.2.

In this example, we assume that P_B and P_C do not have overlapping data regarding suggestions. However, P_A has overlapping data with both P_B and P_C, which means P_A provides success ratings to the suggestions provided by P_B and P_C. This means that there are two different ways how a solution mapping can be obtained to the query in Listing 3.2. Either the data from P_A is joined with the data from P_B, in which case the solution mapping would cost \$0.18 (1 triple from P_A for \$0.10 and 4 triples from P_B for \$0.02), or the data from P_A is joined with the data from P_C, in which case the solution mapping would cost \$0.22 (1 triple from P_A for \$0.10 and 4 triples from P_B for \$0.03).

3.4.2 Host

Hosts operate computers that run SPARQL endpoints for querying data products. They provide the computational and network resources needed to query the providers' data products. Hence, they ensure the reliability, availability, security, and performance, which are usually specified as *Quality of Service* [Dustdar et al., 2012].

Like cloud service providers, hosts incur the fixed cost of operating the infrastructure, possibly some variable cost relative in the size of the data they store, and some marginal cost in form of the computational resources involved for each query they execute. The

host's marginal costs occur whenever the providers' data are queried, independently of whether any data product will eventually get allocated or not. Similar to a Webhost for traditional Web content, the hosts in our market concept have to charge the data providers to cover their costs and make some profit.

Note that a host can store data from multiple data providers and that some data providers may choose to act as their own host. A host has to make sure that nobody can access the data without agreeing to the terms defined by the providers.

3.4.3 Customer and Allocation Rule

A customer is a person, or a program acting on behalf of a person, who has a SPARQL *query* and wants to buy an *allocation* of solution mappings to this query. Depending on the marketplace, the customer might have the choice to use his or her own allocation rule, use one of the marketplace's built-in allocation rules, or use an allocation rule provided by a third party.

The allocation rule sits between the marketplace and the customer. Conceptually, the allocation rule is an independent entity which takes a query answer as input and produces an allocation as output. Practically, the allocation rule can be (1) a part of the marketplace, in which case the user has to provide the marketplace with the necessary parameters for the specific allocation rule to run the allocation process, (2) a part of the the customer, in which case the customer has to inform the marketplace about the chosen allocation, (3) or, an independent entity, in which case this entity has to get the necessary parameters for the specific allocation rule from the customer, has to inform the marketplace about the allocation decision, and has to forward the allocation to the customer once the payments are done. The third option is in particular useful if the allocation process is computationally expensive.

The marketplace will not deliver the full query answer to the allocation rule but will anonymize the query answer such that the allocation process has enough information to choose an allocation. Once the allocation rule informs the marketplace about the chosen allocation and the payments are done, the marketplace will deliver the actual data. In Section 3.5 we will discuss the process of anonymization and in Section 3.6 will discuss the allocation process in detail.

3.4.4 Marketplace

The role of the marketplace is to coordinate the exchanges between the customers posing queries and the hosts serving answers based on the providers' data products. As such it can be seen as an extension of a traditional federated query engine with economic considerations.

The marketplace allows a customer to buy an allocation made from the data providers' triples. For this, the market needs to determine the solution mappings which can be allocated. Based on our previous work in [Grubenmann et al., 2017a], it is unlikely that an allocation based on some data synopses will produce satisfying results for the customer. Hence, the marketplace has to execute the customer's query to create the solution mappings which could be potentially allocated. The marketplace can either run the query

on all, for the query relevant, data products or rely on some source selection and join-prediction service (see [Saleem et al., 2016] for a survey) to preselect a set of the most promising data products.

The customer's payment for an allocation is independent of the query execution. Consequently, the marketplace can optimize the query execution based on traditional federated query optimization techniques without having to consider the prices of the different data products. After the query execution, the customer has to decide which of the obtained solution mappings to buy. Either the customer decides based on an allocation rule or directly chooses a set of solution mappings. This means that the market might execute the query on some data products' triples which might not be included in the customer's allocation, eventually. The customer has to pay the price for all allocated solution mappings to the marketplace, which redirects the money to the respective provider.

As discussed before, the providers will pay the hosts for their services. In addition, the providers also have to pay the marketplace a certain fee to keep it operational. Since the hosts and the marketplace are financially compensated by the providers, the providers will include these payments into their pricing decision. In addition, the market can use part of the generated revenue to subsidize providers which did not get allocated. However, if a provider fails to get allocated over a longer time, the provider's data might simply not be relevant at all, and the market can decide to stop subsidizing such providers. The payment to the host and the payment to the marketplace are transparent to the customer. Hence, the customer's allocation rule has to consider only the prices indicated by the data providers and not any additional payments to the hosts or market. Fig. 3.3 illustrates the money flow between marketplace, providers, hosts, and the customer. The solid arrows indicate payments which are inflicted whenever an allocation is delivered to a customer. The customer pays the marketplace which in turn pays the contributing providers. Note that in Fig. 3.3, only one provider is contributing to the allocation and hence, only this provider is payed. All the providers in the marketplace have to pay service fees to the marketplace and the hosts, indicated by the dashed arrows, which are payed independent of the payment from the customer to the marketplace and to the providers.

3.5 Implementation: FedMark

In this section, we present our implementation of FedMark. FedMark is based on the federated querying engine FedX [Schwarte et al., 2011b]. The core idea of FedMark is that each data product is represented as RDF statements, which describe the RDF data the product contains and any meta-information about the data. This allows one to access the necessary information about all data products as well as their contents with a single, federated SPARQL query. Additionally, it is possible to restrict the query answer including only providers having specific properties by changing the query, accordingly. We will now show how a SPARQL query can be rewritten to (1) extract the necessary additional information about the product and (2) exploit the information about a product to restrict the query answer.

In traditional federated SPARQL query execution, a SPARQL query is split up into subqueries which are sent to different endpoints. Combining the (sub-)query answers from

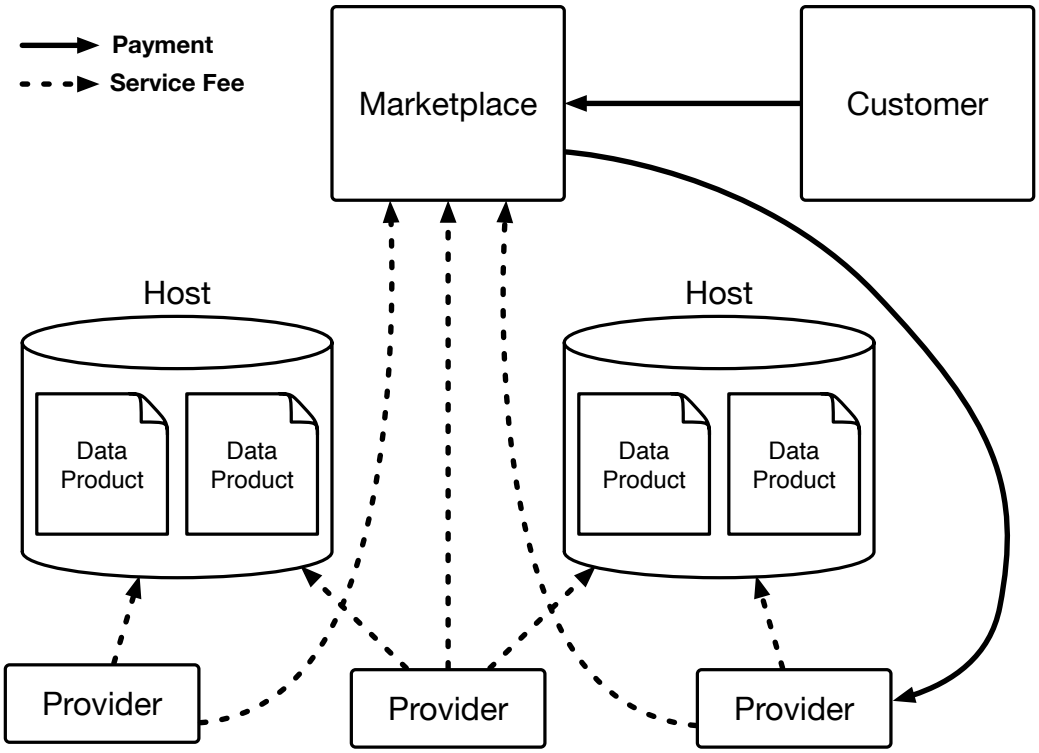


Fig. 3.3: The customer pays the marketplace which forwards the money to the providers. The providers pay the marketplace and the hosts a certain fee for their services.

the different subqueries results in the final query answer. Instead of sending the subqueries directly to the endpoints, FedMark replaces each occurrence of a triple pattern inside a subquery by a more complex graph pattern. This new graph pattern encloses the triple pattern into a `GRAPH` graph pattern. A `GRAPH` graph pattern uses the `GRAPH`-keyword to refer to the (named) RDF-graph which contains the matched triple pattern. FedMark also adds another triple pattern to refer to the product which contains the RDF-graph. Listing 3.3 shows the general form of such a graph pattern. The `?graph` variable will be bound to the name (URI) of the graph which contains the triple matching the original triple pattern. If a product contains a certain RDF graph, this is expressed by the statement `?product market:contains ?graph`, where `?product` will be bound to the URI of the product. By referring to the URI of the product by using the variable `?product`, one can extract further information, e.g. the price, from the data product or restrict the query answer, e.g. by allowing only products having a rating greater than 8.0 (Listing 3.4).

Listing 3.3: A Graph Pattern used for the execution of subqueries in FedMark.

```
GRAPH ?graph {
  [original triple pattern]
```

```
}
?product market:contains ?graph .
```

Listing 3.4: Additional information about a product are extracted and filtered.

```
?product market:price_usd ?price .
?product market:rating ?rating . FILTER (?rating >= 8)
```

Once the query is rewritten as described above, FedMark can execute the new subqueries on the available endpoints and create the query answer. This query answer is the basis on which the allocation rules can now decide which solution mappings to allocate. To prevent revealing the actual data before payment, FedMark does not give the actual query answer to the allocation rule. Instead, FedMark anonymizes the triples used to form the different solution mappings and reveals to the allocation a *summary* which contains (1) the anonymized triples which are needed to form a specific solution mapping and (2) all meta-data available for these anonymized triples.

Table 3.2 shows an example of such a summary. Each row in the left table represents one solution mapping of the query answer. The right table illustrates potential meta-data which could be available for the triples.

Table 3.2: Example Summary for the Allocation Rule

Solution Mapping	Triples
1	t_1, t_2
2	t_1, t_3
3	t_2, t_3

Triple	Price	Quality
t_1	\$0.01	$\star$
t_2	\$0.02	$\star\star\star$
t_3	\$0.03	$\star\star$

3.6 Allocation Rules

As introduced in the last section, a customer can use an *allocation rule* to instruct a program which solution mappings should be allocated and returned. In contrast to a manual selection, an allocation rule allows a customer to automatize the whole process of buying an allocation. This possible automatization is an important aspect of a machine processable WoD, as it allows a customer to instruct a program to buy and process semantic data as needed without any interference from the customer.

The allocation rule is not part of the core concept of FedMark, as our marketplace does not have to know *how* the customer decided for a specific allocation. The only important information is *which* solution mappings are allocated by the customer. Hence, the allocation rule is transparent to FedMark and only its outcome is important. However, implementations of our FedMark concept can provide helpful interfaces and predefined allocation rules to support customers with formulating and implementing an appropriate allocation rule. Eventually, it is the responsibility of the customer to make a good

allocation decision or come up with a good allocation rule to benefit most from the data FedMark can offer.

In the following, we want to present different allocation rules which could be implemented by the customer. As mentioned before, FedMark does not natively provide or constraint the allocation rule. Hence, the presented allocation rules are just a selection of possible allocation rules. We decided to discuss these allocation rules because they illustrate an interesting trade-off between optimality of the allocation and scalability with respect to the number of available solution mappings.

All allocation rules which we will present here have in common that they try to find an allocation which maximizes the customer's *utility*. We assume that the utility is *quasi-linear*. This means that the utility of an allocation is the *value* a customer has for this specific allocation minus the *price* the customer has to pay for it. The customer's value indicates how much the customer is *maximally* willing to pay for a specific allocation.

The price of an allocation is just the sum of the prices for each triple, as indicated by the data providers. The value of an allocation, however, is a private knowledge of the customer and needs to be defined with a function, the *valuation*. The valuation is used by the allocation rule to assert the value of a specific allocation. In the following, we will restrict ourselves to valuations which are *linear* with respect to the solution mappings, this means that the customer's *value* for an allocation is the sum of the values of the solution mappings contained in the allocation. The valuation is used to *discriminate* between different possible allocations containing different solution mappings.

Definition 4 (Linear Valuation). *A* linear valuation *is a linear function* $V : \mathcal{P}(\Omega) \to \mathbb{R}_+$ *that assigns a value to each allocation* $a \subseteq \rho$*. The valuation has the form* $a = \{\omega_1, \ldots, \omega_n\} \mapsto \sum_{i=1}^{n} v(\omega_i)$*, where* $v(\omega_i) =: v_i$ *is the customer's value for solution mapping* ω_i*.*

Definition 5 (Customer's Utility). *The customer's* utility $u \in \mathbb{R}$ *for an allocation* a *is the difference between the customer's value* $v = V(a)$ *of the allocation minus its* payment $\Pi(a)$*, where* $\Pi : \mathcal{P}(\Omega) \to \mathbb{R}_+$ *is a function defined by the marketplace that determines the customer's payment for the allocation.*

In addition to the valuation, the customer has the possibility to add a *budget constraint*. A budget constraint acts as a cap on the payment for the customer and allows the customer control over the maximal amount spent for an allocation.

Returning to our example, we include the customer's valuation:

Example 4. A customer might be willing to pay up to \$0.25 for any solution mapping to the query in Listing 3.2, but is willing to add an additional \$0.10 if all triples originate from a reliable source. In this case, every solution mapping to the query in Listing 3.2 has a value of \$0.25, if at least one of the sources is not considered reliable by the user, and a value of \$0.35, if all sources are considered reliable.

In Section 3.7 we will compare different allocation rules and show under which circumstances which of them should be preferred.

3.6.1 Integer Programming Allocation Rule

The Integer Programming Allocation Rule maximizes the customer's *utility* given a customer's query q, the valuation function $V(\cdot)$, the prices $\pi_1, \ldots, \pi_n$, and the budget constraint s. Hence, the allocation rule describes an optimization problem. We will now show how we can express this optimization problem as an Integer Programming Problem:

Let $\tau_j \in \{0, 1\}$ with $j \in \{1, \ldots, n\}$ be a binary variable indicating whether the triple t_j is bought, π_j the price associated with buying the triple, $r_i \in \{0, 1\}$ with $i \in \{1, \ldots, k\}$ a binary variable indicating whether the solution mapping ω_i can be obtained from the current allocation of triples, where k is the number of all possible solution mappings, and $v_i := v(\omega_i)$ the value for the solution mapping ω_i. Let further s be the budget of the customer which acts as a cap on the total payment.

The objective is to find values for $\tau_1, \ldots, \tau_n$ and $r_1, \ldots, r_k$ which maximizes the utility $u(\tau_1, \ldots, \tau_n, r_1, \ldots, r_k)$, that is the sum of the values of the allocated solution mappings minus the price of the necessary triples:

$$u(\tau_1, \ldots, \tau_n, r_1, \ldots, r_k) = \left(\sum_{j=1}^{k} r_j \cdot v_j \right) - \left(\sum_{i=1}^{n} \tau_i \cdot \pi_i \right) \tag{1}$$

In addition to the objective, we also have constraints which have to be respected by the solution of the Integer Programming Problem.

The first constraint is that for a solution mapping $\omega_j \in \Omega$ all the $n_j = |I_j|$ relevant triples $\{t_i \mid i \in I_j\}$ have to be allocated to include that solution mapping into the allocation, where I_j is the *index set* of the indices of the relevant triples. This means that the binary variable r_j can only be set to one when all the variables τ_i with $i \in I_j$ are set to one. We can enforce this with the following linear constraint:

$$\sum_{i \in I_j} \tau_i - n_j \cdot r_j \geq 0 \tag{2}$$

It is possible that multiple data products offer the same triple, in this case, only the cheapest triple will be considered. If multiple data products offer the same triple at the same price, one of the triples is randomly chosen as the relevant triple.

The second constraint is that the price of the allocation does not exceed the budget s:

$$\sum_{i=1}^{n} \tau_i \cdot \pi_i \leq s \tag{3}$$

Equations 4 show the general form of the Integer Program for this optimization problem:

$$\begin{aligned} \text{Objective: } & \max \left(\sum_{j=1}^{k} r_j \cdot v_j - \sum_{i=1}^{n} \tau_i \cdot \pi_i \right) \\ \text{Subject to: } & \sum_{i \in I_1} \tau_i - n_1 \cdot r_1 \geq 0 \\ & \vdots \\ & \sum_{i \in I_k} \tau_i - n_k \cdot r_k \geq 0 \\ & \sum_{i=1}^{n} \tau_i \cdot \pi_i \leq s \\ \text{Bounds: } & r_1, \ldots, r_k \in \{0, 1\} \\ & \tau_1, \ldots, \tau_n \in \{0, 1\} \end{aligned} \tag{4}$$

In Example 5 we show how such an Integer Programming Problem for our scenario could look like.

Example 5. We extend Example 4 by assuming that provider P_A can contribute triples $t_1, \ldots, t_5$, provider P_B triples $t_6, \ldots, t_{17}$, and provider P_C triples $t_{18}, \ldots, t_{25}$. Further, assume that the query answer consists of the solution mappings with their respective value according to Table 3.3.

Table 3.3: Solution mappings, their required triples, and the value.

Solution Mapping	Triples	Value
ρ_1	t_1, t_6, t_7, t_8, t_9	\$0.25
ρ_2	$t_2, t_{10}, t_{11}, t_{12}, t_{14}$	\$0.35
ρ_3	$t_3, t_{14}, t_{15}, t_{16}, t_{17}$	\$0.35
ρ_4	$t_4, t_{18}, t_{19}, t_{20}, t_{21}$	\$0.35
ρ_5	$t_5, t_{22}, t_{23}, t_{24}, t_{25}$	\$0.25

In this case, the Integer Programming Problem has the following form:

$$
\begin{aligned}
\text{Objective:} \quad & \max(\$0.25 \cdot (r_1 + r_5) \\
& + \$0.35 \cdot (r_2 + r_3 + r_4) \\
& - \$0.10 \cdot (\tau_1 + \cdots + \tau_5) \\
& - \$0.02 \cdot (\tau_6 + \cdots + \tau_{17}) \\
& - \$0.03 \cdot (\tau_{18} + \cdots + \tau_{25})) \\
\text{Subject to:} \quad & \tau_1 + \tau_6 + \tau_7 + \tau_8 + \tau_9 - 5 \cdot r_1 \geq 0 \\
& \tau_2 + \tau_{10} + \tau_{11} + \tau_{12} + \tau_{13} - 5 \cdot r_2 \geq 0 \\
& \tau_3 + \tau_{14} + \tau_{15} + \tau_{16} + \tau_{17} - 5 \cdot r_3 \geq 0 \\
& \tau_4 + \tau_{18} + \tau_{19} + \tau_{20} + \tau_{21} - 5 \cdot r_4 \geq 0 \\
& \tau_5 + \tau_{22} + \tau_{23} + \tau_{24} + \tau_{25} - 5 \cdot r_5 \geq 0 \\
& \$0.10 \cdot (\tau_1 + \cdots + \tau_5) \\
& + \$0.02 \cdot (\tau_6 + \cdots + \tau_{17}) \\
& + \$0.03 \cdot (\tau_{18} + \cdots + \tau_{25}) \leq \$0.65 \\
\text{Bounds:} \quad & r_1, \ldots, r_5 \in \{0, 1\} \\
& \tau_1, \ldots, \tau_{25} \in \{0, 1\}
\end{aligned}
\tag{5}
$$

The two big advantages of the Integer Programming Allocation Rule are that (1) it can be solved using standard optimization tools which are specialized in such problems and (2) the allocation found by this rule is optimal: a customer cannot gain more utility by any other allocation, given the valuation and prices. However, the allocation rule also has also a disadvantage: Solving the Integer Programming Problem is NP-hard, which means that the Integer Programming Allocation Rule has a limited scalability. We will investigate the runtime needed to solve the problem in Section 3.7. Another drawback of this allocation rule is that it is limited to a linear valuation of the allocation. As soon as the value for a single solution mapping is not constant—this can happen for example if the customer has a decreasing marginal value for the solution mappings—the optimization cannot anymore be formulated as an Integer Programming Problem and the solving tools cannot be used. Given this drawback, we present another allocation rule which will complement the Integer Programming Allocation Rule.

3.6.2 Greedy Allocation Rule

The Greedy Allocation Rule tries, as the name suggests, to find a good allocation in a greedy fashion. Hence, the allocation rule does not guarantee that the found allocation is optimal in the sense that it maximizes the customer's utility. The upside of this allocation rule is that (1) it scales better with increasing number of solution mappings which can be allocated, which means it remains feasible in situations where the NP-hard Integer Programming Allocation Rule is not anymore feasible, and (2) the allocation rule is com-

patible with any valuation which is monotonic decreasing. Note that the valuation does not have to be strictly monotonic decreasing but can also be linear.

The idea of the Greedy Allocation Rule is quite simple: Choose the non-allocated solution mapping with the highest *ratio between utility and price* and allocate it, as long as the utility is positive (the value is higher than the price) and the sum of the price of all allocated solution mappings is smaller than or equal to the budget. Whenever a new solution mapping is allocated, the utility of the remaining non-allocated solution mappings must be updated. This is because an allocated solution mapping might include some triples of the non-allocated solution mappings, in which case the specific triples do not have to be bought again and the price for the respective solution mappings decreases.

Algorithm 3.1 shows how the Greedy Allocation rule can be implemented. First, the algorithm initializes the set I with the indices of all solution mappings, the set T_{buy} for the indices of all triples which need to be bought, the allocation a, and the total price Π of allocation a (Line 1–4). Then, the algorithm enters a loop and determines the solution mapping with the highest ratio between utility and price. For this, the algorithm has to get the indices of those triples which are relevant for a specific solution mapping ω_i. This information is provided by the function *relevantTriplesIndices* (Line 7). Afterwards, the indices of those triples which are already considered for buying are removed from T_{buy} (Line 8), because the same triple does not need to be bought twice by the customer. Using the indices of the required triples, the price Π_i (Line 9), the utility u_i (Line 10), and the ratio r_i (Line 11) can be calculated. With this information, the algorithm can determine the index max of the solution mapping with the highest ratio having a price which is still in the budget b and a utility of at least 0 (Line 12). If there is still a solution mapping that reaches these conditions (Line 13), the total price Π, the allocation a, the index set I with all available solution mappings for allocation, and the index set of all triples to be bought T_{buy} are updated accordingly (Line 14–17). If there is no suitable solution mapping or simply no solution mapping at all left, the algorithm stops the loop and returns the allocation a and the total price Π (Line 18–19).

The allocation rule runs in $O(n^2 \log(n))$ time, where n is the number of solution mappings available: For each (at most n) new allocated solution mapping, the algorithm has to sort the remaining non-allocated solution mappings ($O(n \log(n))$) to determine the one with the highest utility.

3.7 Evaluation

To empirically compare the presented allocation rules, we measure runtime and utility for different queries. The first measure, runtime, is chosen to establish that the use of our proposed method is *feasible* and practical in a WoD setting from a computational point of view. The second metric, utility, indicates that the results are *desirable* and how close the greedy rule approaches the optimal.

Algorithm 3.1 Algorithm for the Greedy Allocation Rule.

Data: Solutions $\omega_1, \ldots, \omega_n$, values $v_1, \ldots, v_n$, prices $\pi_1, \ldots, \pi_k$, budget b
Result: Allocation a, Payment Π

$I \leftarrow \{1, \ldots, n\}$
$T_{\text{buy}} \leftarrow \emptyset$
$a \leftarrow \emptyset$
$\Pi \leftarrow 0$
do
  for $i \in I$ **do**
    $T_{\text{relevant}} \leftarrow \text{relevantTriplesIndices}(\omega_i)$
    $T_{\text{required}} \leftarrow T_{\text{relevant}} \setminus T_{\text{buy}}$
    $\Pi_i \leftarrow \sum_{j \in T_{\text{required}}} \pi_j$
    $u_i \leftarrow v_i - \Pi_i$
    $r_i \leftarrow \frac{u_i}{\Pi_i}$
  $max \leftarrow \text{argmax}_{i \in \{1,\ldots,n\}}(\{r_i | \Pi + \Pi_i \leq b \text{ and } u_i \geq 0\})$
  if $\exists max$ **then**
    $\Pi \leftarrow \Pi + \Pi_{max}$
    $a \leftarrow a \cup \{\omega_{max}\}$
    $I \leftarrow I \setminus \{i\}$
    $T_{\text{buy}} \leftarrow T_{\text{buy}} \cup T_{\text{relevant}}$
while $\exists max$ *and* $I \neq \emptyset$
return a, Π

Table 3.4: Number of solution mappings and triple per solution mapping.

Query	Solution Mappings	Triples per Solution Mapping
CD1	90	1–2
CD6	11	4
LD1	308	3
LD2	185	3
LD3	159	4
LD4	50	5
LD5	28	3
LD6	39	5
LD7	1216	2
LD8	22	5
LD11	376	5
LS1	1159	1
LS2	333	1–2
LS3	9054	5
LS5	393	6
LS6	28	5
LS7	144	4–5

3.7.1 Evaluation Setup

We use two different scenarios: one based on the FedBench benchmark and a new, synthetic scenario.

FedBench Scenario: The goal of this first scenario is to evaluate our procedure in a well-established realistic setting. FedBench [Schmidt et al., 2011] consists of 9 datasets on various topics and 25 SPARQL queries. We excluded queries with only 1, 2, or 3 solution mappings, as their allocation would be rather trivial. This left us with 17 queries. The number of solution mappings range from 11 to 9054 per query. Table 3.4 shows the number of solution mappings and triples per solution mapping for each of the 17 selected FedBench queries.

Synthetic Scenario: The goal of this second scenario is to evaluate the scaling behavior of the allocation procedure whilst varying both the number of solution mappings per query answer and the number of unique triples contained therein. To that end we generated hypothetical queries that have randomly generated query answers (as we only require the answer sets for evaluating the allocation procedure). The number of solution mappings s per answer varies between 50 and 1000. Each solution mapping consists of 5 triples, hence we have $n = 5s$ triples in an answer. To simulate the *diversity* of an answer, we introduce the parameter $d \in [0, 1]$, which specifies how many unique triples are contained in a query answer. Next, we randomly assign each triple one of $n_{\text{unique}} = 1+d\cdot(n-1)$ identifiers. The result is an answer set in which all triples are the same for $d = 0$ and each triple in the query answer is unique for $d = 1$. This procedure generates query answers of varying size s and number of unique triples n_{unique}.

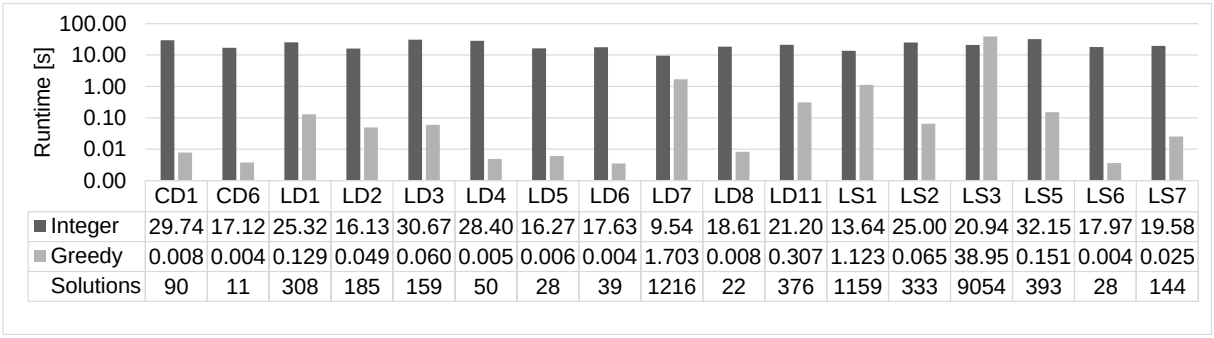
Parameters: We set the price for each triple to a uniformly distributed random number between \$0.00 and \$1.00. The number of triples per solution mapping multiplied by a uniformly distributed random number between \$1.00 and \$2.00 gives us the value for each solution mapping. The budget for each query is set such that only 50% of the solution mappings can be obtained. Using these numbers ensures that (1) there is at least one affordable allocation having positive utility and (2) not all solution mappings can be allocated. This guarantees that the allocation problem does not become trivial to solve.

3.7.2 Results

We discuss both scenarios in turn.

FedBench Scenario: Fig. 3.4 graphs the execution time for the 17 selected FedBench queries for the Integer Rule and the Greedy Rule in seconds. It shows that the Integer Allocation Rule is by orders magnitude slower than the Greedy Allocation Rule for most queries. One exception is query LS3, which actually has a longer runtime for the Greedy Rule. LS3 is also the query with the highest number of solution mappings. One explanation is the high diversity in a large number of solution mappings that benefits the integer approach.

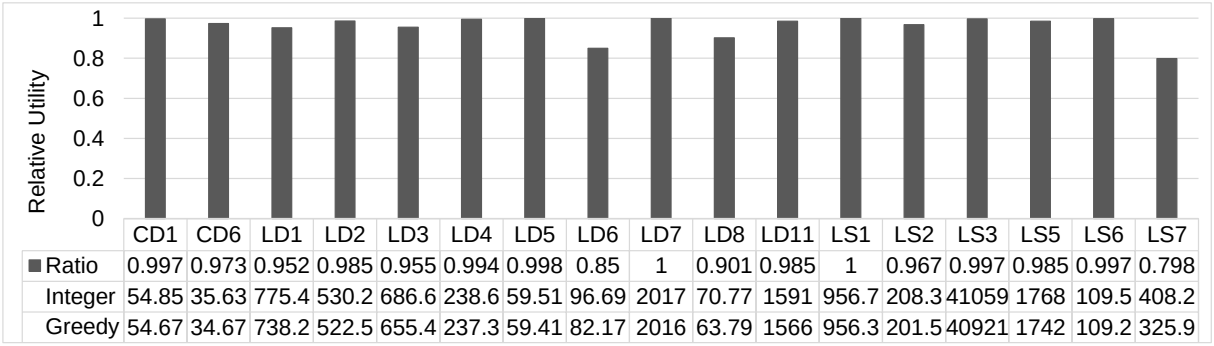
The ratio between the utility of the Greedy Rule and the Integer Rule for the 17 selected FedBench queries is graphed in Fig. 3.5. The graph shows that the Greedy Rule has a utility which is very close to the Integer Rule, which maximizes utility given the prices and values. The evaluation shows that for the FedBench queries, the Greedy Rule provides allocations of comparable quality to the Integer Rule in orders of magnitude smaller time.



	CD1	CD6	LD1	LD2	LD3	LD4	LD5	LD6	LD7	LD8	LD11	LS1	LS2	LS3	LS5	LS6	LS7
Integer	29.74	17.12	25.32	16.13	30.67	28.40	16.27	17.63	9.54	18.61	21.20	13.64	25.00	20.94	32.15	17.97	19.58
Greedy	0.008	0.004	0.129	0.049	0.060	0.005	0.006	0.004	1.703	0.008	0.307	1.123	0.065	38.95	0.151	0.004	0.025
Solutions	90	11	308	185	159	50	28	39	1216	22	376	1159	333	9054	393	28	144

Fig. 3.4: Runtime in seconds for the Integer Programming Allocation Rule (Integer) and the Greedy Allocation Rule (Greedy) for the FedBench benchmark.

Synthetic Scenario: We will now focus on the scaling behavior of both allocation rules. Fig. 3.6 shows how the Integer Programming Allocation Rule scales with respect to the diversity d for different number of solution mappings. Note that the runtime is plotted in a logarithmic scale and that we plot with respect to d and not n_{unique}, as the latter is dependent on s. For this evaluation, we used our own synthetic data as described



	CD1	CD6	LD1	LD2	LD3	LD4	LD5	LD6	LD7	LD8	LD11	LS1	LS2	LS3	LS5	LS6	LS7
Ratio	0.997	0.973	0.952	0.985	0.955	0.994	0.998	0.85	1	0.901	0.985	1	0.967	0.997	0.985	0.997	0.798
Integer	54.85	35.63	775.4	530.2	686.6	238.6	59.51	96.69	2017	70.77	1591	956.7	208.3	41059	1768	109.5	408.2
Greedy	54.67	34.67	738.2	522.5	655.4	237.3	59.41	82.17	2016	63.79	1566	956.3	201.5	40921	1742	109.2	325.9

Fig. 3.5: Ratio between the utility of the Greedy Allocation Rule (Greedy) and the Integer Programming Allocation Rule (Integer) for the FedBench benchmark.

above. For some plots, the graph has some missing points. The missing points indicate parameter combinations that did not yield results within 12 hours of optimization. As the figure shows, the runtime complexity explodes if the diversity is in the lower third of the spectrum and the number of solution mappings is high enough. For a diversity of 0, the allocation problem becomes trivial as there is only one triple which can be chosen. For a diversity of 1, the different solution mappings in a query answer are *independent*, meaning that they do not share any triples. Also, in this case, the allocation problem seems to be simpler to solve, although not as simple as when the diversity is 0. For a diversity in between 0 and 1, the allocation problem becomes harder. This is because the solution mappings are now more dependent on each other because they share triples: some combinations of solution mappings will be much cheaper than other combinations, because they can exploit the fact that they share some triples and their prices. Interestingly, the less the diversity is, the more time the Integer Programming Allocation Rule needs to solve the allocation problem. At least, until the diversity gets close to 0, at which point the number of triples is very low and the allocation problem becomes much easier.

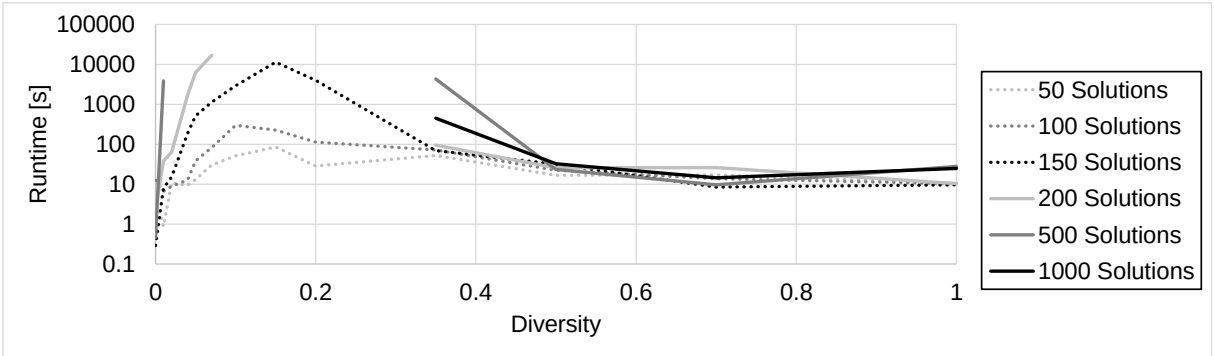


Fig. 3.6: Runtime in seconds (on a logarithmic scale) for the Integer Programming Allocation Rule for different diversities and query answer sizes.

The scaling behavior of the Greedy Allocation Rule with respect to the diversity d for different number of solution mappings is shown in Fig. 3.7. The plot uses the same scale as in Fig. 3.6 to make it easier to compare the results. As Fig. 3.7 indicates, the runtime for the Greedy Allocation Rule does not suffer from the same explosion of runtime as the Integer Programming Allocation Rule when the diversity is low. The reason for this is quite simple: the Greedy Rule does have to consider which combination of solution mapping could exploit the overlap of triples the most. Instead, the allocation rule just chooses the next best solution mapping and updates the prices, accordingly. Nevertheless, we can observe some trend that the runtime is higher for lower diversity than it is for high diversities or a diversity of 0. This can be explained by considering how much the Greedy Allocation Rule has to resort the solution mappings after each step. If the diversity is high, there are only few solution mappings for which the price changes after a solution mapping is selected. Hence, the resorting can be done faster. If the diversity is low, however, selecting a solution mapping does impact more remaining solution mappings, due to the increased overlap. Hence, the resorting takes more time. Eventually, for a diversity of 0, after selecting the first solution mapping all other solution mappingss have a price of 0 (because they all need the same triple which is already bought), which means that no resorting at all is needed.

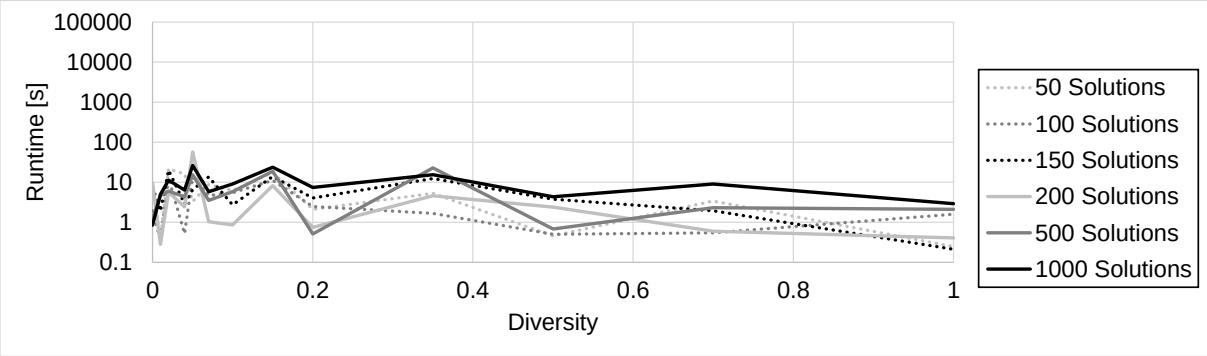


Fig. 3.7: Runtime in seconds (on a logarithmic scale) for the Greedy Allocation Rule for different diversities and query answer sizes.

As our evaluation shows, the runtime of the Integer Rule depends highly on the diversity of the triples within a query answer. Hence, even a query answer with a lot of solution mappings can be feasible for the Integer Rule whereas another query answer with fewer solution mappings might not be feasible. The Greedy Rule behaves more stable with changing diversity. Paired with our observations about the utility of resulting solution mappings in the FedBench scenario, we can infer that the Greedy approach seems to provide good allocations within reasonable time bounds for realistic scenarios.

3.8 Limitations and Conclusions

To grow further and be able to serve as a high-quality data source, the WoD has to find the means to fund the creation, serving, and maintenance of data sources. In this paper, we proposed a new paradigm for funding these activities in the form of a market for data that combines a market-based approach with principles of federated querying. We presented FedMark, a prototype that implements the concepts we introduced in this paper. In addition, we introduced two possible allocation rules which can be used by a customer to decide for a specific allocation. As we have seen, both allocation rules have different properties with respect to runtime and utility. While the Integer Allocation Rule guarantees an optimal allocation with respect to utility, the runtime of this rule can exceed any reasonable time limit under certain condition, as we have seen in the evaluation section. The Greedy Allocation Rule does not suffer from extensive runtimes in the scenarios we investigated. However, the Greedy Rule cannot guarantee an optimal allocation. Although, we have seen that the utility is often very close to an optimal allocation. In practice, a customer would be advised to run both allocation rules in parallel and specify a time out. After the time runs out, the customer can check whether the Integer Allocation Rule has found an optimal allocation. If not, the current best solution of the Integer Allocation Rule can be compared to the outcome of the Greedy Rule. Whichever rule produces the best result under the time constraints should be picked by the customer.

Another advantage of the Greedy Rule is that it can handle decreasing marginal values for the solutions. If the customer's valuation is not linear, he might not have any other choice but to use the Greedy Rule. Obviously, we need to explore a set of other allocation rules to understand the trade-offs for the various different valuation needs of the customer.

In addition, we need to revisit our assumptions and check if they are truly realistic. As an example, consider the assumption that providers can amortize their fixed costs over many transactions. This is only true if their goods are actually sufficiently attractive to be bought, which again depends on the competitiveness of the marketplace. Whilst we believe that this is true for many data products (e.g., financial data) we will have to investigate where this assumption does not hold. Second, this paper did not discuss how a provider decides on the optimal pricing. Whilst we did run an analysis indicating that it is favorable for a provider to learn the price, we did not evaluate how well that price *can* be learned—a task for future work. Third, we need to explore the possibility of selling query subscriptions, which opens the way to mechanisms akin to the ones that are currently dominating the entertainment industry. Fourth, we need to explore market-aware optimizations for FedMark and evaluate their influence on the speed of query execution. Finally, the generalizability of our evaluation might be hampered by the use of FedBench. Indeed, FedBench's limitations led us to run a second evaluation with synthetic data. Whilst an evaluation in additional real-world scenarios is desirable and should be subject of future work, we believe that our evaluation highlights the key properties of our allocation rules and, hence, establishes their applicability.

Whatever the shortcomings of FedMark and our concept, we believe that the contributions of this paper are a first step in the principled exploration of a financially stable and, therefore, sustainable Web of Data.

Chapter 4

Finance Data through Sponsors

This chapter is based on:

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. 2018. ***Financing the Web of Data with Delayed-Answer Auctions****. In WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186002*

Financing the Web of Data with Delayed-Answer Auctions

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken

Department of Informatics, University of Zurich, Switzerland

Abstract. The World Wide Web is a massive network of interlinked documents. One of the reasons the World Wide Web is so successful is the fact that most content is available free of any charge. Inspired by the success of the World Wide Web, the Web of Data applies the same strategy of interlinking to data. To this point, most of data in the Web of Data is also free of charge. The fact that the data is freely available raises the question of financing these services, however. As we will discuss in this paper, advertisement and donations cannot easily be applied to this new setting.
To create incentives to subsidize data providers, we propose that sponsors should pay the providers to promote sponsored data. In return, sponsored data will be privileged over non-sponsored data. Since it is not possible to enforce a certain ordering on the data the user will receive, we propose to split up the data into different batches and deliver these batches with different delays. In this way, we can privilege sponsored data without withholding any non-sponsored data from the user.
In this paper, we introduce a new concept of a delayed-answer auction, where sponsors can pay to prioritize their data. We introduce a new model which captures the particular situation when a user access data in the Web of Data. We show how the weighted Vickrey-Clarke-Groves auction mechanism can be applied to our scenario and we discuss how certain parameters can influence the nature of our auction. With our new concept, we build a first step to a free yet financial sustainable Web of Data.

4.1 Introduction

The Web of Data (WoD) is the result of applying the principle of interlinking documents, which fueled to growth of the World Wide Web, to data, which results in so called *Linked Data*. Like in its predecessor, the WWW, most content of the WoD is to date free of any charge. Allowing users to access Linked Data for free introduces new challenges when it comes to financing these services. The question we are answering in this paper is: *How can the WoD be free and financially sustainable at the same time*?

The World Wide Web is not only the inspiration of the Web of Data but also serves as an example of how to finance such services. Hence, it is natural to think about applying the techniques which finance the WWW to the WoD. One of the biggest financial motors of the WWW is advertisement. What made advertisement in the WWW more efficient than in other medias is the fact that advertisement in the WWW is customized to the user. Hence, it is easier for advertisers to target a specific user group. This customization is mainly achieved either by showing an ad based on a keyword entered in a search page or by embedding ads which are related to a certain webpage's content. Unfortunately, these techniques do not apply that easily to the WoD. The main problem of the WoD with respect to advertisement is that the data provider has no influence on how the content is presented to the user: In the WWW, advertisement can be added to the presentation of the content at the discretion of the data provider. In the WoD, however, a user receives the data in a structured format. This structured format allows algorithms to automatically

process the data further, if needed, and does not provide the means to add any additional advertisements. Accessing the WoD from a user's perspective is more similar to accessing a database than accessing a website's content.

Donations could be an alternative to financing the WoD. Wikipedia is an example of a content provider in the WWW which is financed by donations. However, a significant part of the users is not aware that there is a possibility to donate to Wikipedia or do not know how to donate [Glott et al., 2010]. Such a lack of *awareness* for the need of donations will be even more pronounced in the WoD. Users in the WoD often access a lot of datasets at the same time. Sometimes these datasets are part of a federation and can be accessed transparently through a single accessing point. Very often, a user might simply not be aware of using a certain dataset let alone realizing that the provider requires financial support through donations.

To remedy this situation, we propose that *sponsors* who are interested in promoting certain data will subsidize *all data provider which are involved in creating the specific solution containing the subsidized data*. A sponsor can be anybody who gains an advantage if certain data is distributed to as much users as possible. However, this means that we need a way to *privilege* highly sponsored over less sponsored or non-sponsored data. In the WWW, search engines, for example, can change the ordering of search results to privilege certain websites and order advertisements on the search result page based on the payments. In the WoD, the query language SPARQL [Harris and Seaborne, 2013] which is used to query the desired data allows a user to specify the ordering of the received data. Even if we disable such a functionality, the structured format in which the data is delivered easily allows a user, or the program which is querying on behalf the user, to reorder the received data ad libitum. This means that it is not possible to force a certain *ranking* of the received data upon the user within a single query answer. One alternative is to simply deliver only part of the available data and withhold the rest. Data would then be delivered or not based on the amount of sponsorship the data received. This would create a situation where different sponsors compete to be part of the delivered data. Whereas such a situation would create enough competition between sponsors to ensure a certain revenue, it opposes the idea that *all* data should be freely available. Hence, we propose a new concept where the requested data is *delayed* depending on how much a certain sponsor is willing to pay for its data being privileged over the data of other sponsors. Our new concept is aligned with the idea that all data is freely available but creates at the same time an incentive for sponsors to promote certain data. The revenue generated by our concepts can be used to finance the providers which are responsible for hosting and maintaining the data.

By delaying part of the data requested by a user, we are harming the user's experience to a certain degree. Obviously, a user would prefer to get all requested data immediately instead of receiving the data in consecutively delayed batches. We discuss the user's experience in detail in Section 4.6.3. In addition, a user would prefer to receive data in an unbiased way instead of receiving first the data which received the highest sponsoring. To remedy this problem, we introduce an extension to our model in Section 4.7 where part of the data is randomly assigned to batches.

The contributions of our paper are: (1) we introduce our new concept of a *delayed-answer auction* where sponsors can pay to have sponsored data delivered quicker to the user, (2) we introduce a new link selection model (akin to click models in the WWW) which captures the probability that a user selects a link contained in a certain solution, (3) we show how the weighted VCG auction mechanism can be applied to our scenario and we discuss how certain parameters of the auction can influence the nature of our auction, and (4) we discuss an extension to our model where only part of the data delivery is influenced by the sponsors.

4.2 Motivation

Imagine a user who wants to make a reservation for a restaurant in Zurich, having a rating of at least 8.0, and which is offering traditional food. Such a user can use the query language SPARQL [Harris and Seaborne, 2013] to express the exact needs. Compared to a keyword based searched in the WWW, such a *semantic search* is much more precise when it comes to expressing which data the user actually wants. Listing 4.1 shows how a SPARQL query for our example could look like.

Listing 4.1: A query which asks for traditional restaurants in Zurich with a ranking of at least 8.0.

```
SELECT ?name ?link
WHERE {
        ?restaurant :City ex:Zurich .
        ?restaurant :Food_Style :Traditional .
        ?restaurant :Reservation_Link ?link .
        ?restuarant :ranking ?rank . FILTER(?rank >= 8) }
```

The user issuing the SPARQL query from Listing 4.1 has an incentive to make a reservation for a restaurant with the desired properties as quick and straightforward as possible. Meanwhile, the different restaurants have an incentive that the user makes the reservation at their own restaurant and not at the competition. Assume now that the different solutions for the query arrive at the user with different delays, e.g., the first solution will appear immediately, the second solution after a few seconds, the third solution even a few seconds later, and so on. It is not hard to imagine that the user would not wait an hour for all thousands of solutions to appear. Instead, a user would probably only wait a few seconds before picking one of the already available solutions and make the reservation. Indeed, as [Lohr, 2012] implied, users become very inpatient over time and even a delay of 250ms can influence a user to visit a website of a close competitor. Similar, [Hamilton, 2009] argue how important latency is for the user's experience of a website. An experiment at Amazon showed that a delay of 100ms resulted in sale decrease of 1% [Kohavi et al., 2009]. Also, an experiment at Google showed that if the time to display search results is increased by 500ms, the revenue is reduced by 20% [Kohavi et al., 2009]. Finally, experiments at Microsoft Live Search showed that when the research result page

is slowed down by a second, ad clicks per user decline by 1.5%, and even by 4.4% when the delay is two seconds [Kohavi et al., 2009].

Based on these findings, we developed a model for delaying solutions for SPARQL query answers where *sponsors* can pay money to prioritize certain solutions and delay others. Using this strategy of delaying part of the query answer, we manage to discriminate different solutions. Similar to a user clicking on a link in the WWW, a user (or program acting on behalf of a user) can decide to look up a certain URI contained in a solution for a query answer. A sponsor is motivated to pay money whenever such a URI lookup directs the user to some service. In our example, the offered service could be a reservation system for the respective restaurant. We will denote URIs which direct a user to some service as *service link*, or just *link*, to distinguish it from other URIs. Depending on how much a sponsor is willing to pay for a visited service link, a solution containing that link might be more or less prioritized. If a user looks up a service link, all the data provider involved in creating the specific solution will receive a share of the money the sponsor is paying. In particular, the provider offering the rankings of different restaurants gets paid for its service.

In Section 4.4 we discuss our concept of delaying solutions in more detail and in Section 4.5 we introduce a formal model for our concept. Note that our model is targeted at query answers for which a delay influences the likelihood that the user will consider a certain solution. This is not always the case. For some queries, a user does not mind waiting a long time receiving an answer. Hence, it is important to keep in mind that our concept is specifically designed for situations were such delays do matter.

4.3 Related Work

Auctions for Sponsored Search Results: Before Google introduced the Generalized Second Price (GSP) auction in 2002, first price models were used for selling ads on search result pages [Wilkens et al., 2017]. The idea of the GSP auction is that each bidder submits one bid, which indicates the bidders value for a click. Different positions (or slots) on the search result page will have different click-through rates. Hence, the bidders value per click translates into a value per slot. Depending on the auctioneer, different ads might have different click-through rates for the same slot, adding an additional layer of complexity. In the simplest variation of the GSP auction, the first slot is given to the bidder declaring the highest value, the second slot to the bidder with the next highest value, and so on. What gives the name to the GSP auction is the fact that each bidder pays a price which equals the value of the next lowest bidder, also known as the *second price*. [Edelman et al., 2007]

The advantage of the GSP auction over first price models is that the GSP auction prevents "cycling" patterns, a situation where prices gradually rise until a sudden drop occurs and the pattern starts over [Edelman and Ostrovsky, 2007]. Shortly after Google implemented the GSP auction, one of their engineers realized that a Vickrey-Clarke-Groves (VCG) auction [Clarke, 1971, Groves, 1973, Vickrey, 1961] could also be implemented to sell ad positions on their search results pages. The advantage of the VCG auction is that

it is truthful, which means that each bidder has an incentive to report their true value as bid. Google refrained from replacing the GSP with the VCG auction because the GSP auction was already growing attention and there would have been additional effort involved, both on the side of Google and the sellers, to change to the VCG auction. [Varian and Harris, 2014]

In [Aggarwal et al., 2007], Aggarwal et al. propose a position-based auctions mechanism where bidders can impose additional constraints on the positions they want their ad to appear. The motivation behind this auction format is that a bidder might have a value if the ad just appears at the top of a search result page, even if the user does not click the ad, because the mere appearance of the ad increases the visibility of the brand.

In the WoD setting, it is unclear which of these auctions are applicable. In section 4.5.3 we will show how the VCG auction mechanism can be applied to our setting.

Click Models for Web Search and Sponsored Search Auctions: Both web search and sponsored search auctions need click models for estimating how likely a user would select a certain search result or a certain ad. Hence, the different models used in these two fields are often overlapping and can often be used in the other field.

The simplest models for user clicks on advertisements is based on the assumption that click-through rates can be separated into two factors: one factor which is only influenced by the ad which wins the slot and one factor which is only influenced by the slot position itself. The result of this assumption, called the *separability assumption*, is that the click-through rate of a certain ad winning a certain slot can be computed by just multiplying these two factors, without having to consider which ads are winning all the other slots. In settings where the click-through rates are not separable, a weighted VCG mechanism may not apply. [Aggarwal et al., 2006]

The click model of [Richardson et al., 2007] provides one example of such separable factors: the influence of the slot is described as the probability that a user sees an ad at a specific position. The influence of the ad is described as the probability that the user clicks on the ad given that the user saw the ad. Note that it is assumed that those probabilities are independent of the ads already shown to the user.

Aggarwal et al. [Aggarwal et al., 2008] introduce a Markovian user model where the user scans a list of ads and makes a decision whether to (1) click on the ad, (2) continue scanning the list, or (3) abort the inspections of the ads. Note that in this model, the probability that a user clicks on a certain ad does not only depend on the ad itself and the position but also on the ads placed in higher slots. Hence, the separability assumption does not hold anymore in this model. A consequence of this is that the GSP allocation of ads is not anymore the most efficient allocation [Aggarwal et al., 2008]. Similar to the model of [Aggarwal et al., 2008], [Craswell et al., 2008] introduces a cascading model for search results where each document is either clicked or skipped. In the latter case, the user continues the scanning of the list of results.

One drawback of the models of [Aggarwal et al., 2008] and [Craswell et al., 2008] is that it is assumed that a user will only continue scanning the ads/search results if no previous ad/search result has been clicked, yet, and hence, assuming that the user will click at most on one ad/search result during the scanning process. [Guo et al., 2009]

introduced the dependent click model which extends the cascading model by introducing conditional probabilities of a user to continue scanning the list of results depending on whether a click occurred on the current document or not and hence, allowing for multiple clicks within a single scanning process.

In [Zhu et al., 2010], Zhu et al. introduce their *General Click Model* which is a more general model for user clicks. Most existing models can be considered a special case of their general model and they showed, for example, that the models of [Richardson et al., 2007] and [Craswell et al., 2008] can be modelled using their General Click Model.

As described in Section 4.2, a user (or program) can visit certain service links, which are URIs which direct the user to some service. Selecting such a service link is very similar to a user clicking on a link in the WWW. However, despite the variety of click models for web search and sponsored search auctions, none of them captures the situation when a user selects a service link from a set of solutions for a SPARQL query. Hence, in Section 4.5.1 we introduce a new selection model which we designed especially for SPARQL query answers.

4.4 Delayed-Answer Auction

In this section, we introduce our concept of a *delayed-answer auction*. At the core of our concept lies the ability of *sponsors* to pay money if a user follows a certain service link contained in a query solution. The data is accessible in form of a SPARQL endpoint. Our auction mechanism makes sure that solutions containing links with higher bids appear with a smaller delay than solutions containing links with lower bids. Hence, our auction mechanism creates a ranking of the solutions by introducing different delays for them.

The user who poses the SPARQL query and the data providers which offer the data needed to answer the query define the context of our auction. Given this context, the sponsors are the participants (or bidder) in the auction. Hence, we will use the expressions *sponsor* and *bidder* interchangeably. The bidders place a *bid* on a specific link contained in a query solution. This bid indicates how much the sponsor is willing to pay if the user visits a certain service link. We call a service link which has a bid placed on it a *sponsored link*.

In our concept, we make the simplifying assumption that each solution contains at most one sponsored link. This means that each solution can be associated with one sponsor and one bid. If nobody is bidding on a link contained in a solution or the solution contains no link at all, the bid is set to 0. We discuss the more general case when there might be multiple sponsored links in a single solution in the limitations in Section 4.8. If a user visits a link contained in a solution for a query answer, the visit of the service link is registered at the auctioneer and the placed bid is charged to the respective bidder. The revenue generated by the auction is used to finance all the entities involved in providing the data which were needed to create the solutions for the query answer.

Figure 4.1 illustrates the process of our delayed-answer auction concept. A user submits a query to the auction. The auction executes the query and generates the query answer. Parts of the query answer get delayed, depending on how much the sponsors bid on certain

links contained in the solutions. Figure 4.2 illustrates how the money gets redistributed by the auction to the providers which were involved in creating the query answer. The service link which is visited directs the user to the auction which, in turn, redirects the user to the appropriate location. This redirecting mechanism is important to keep track of which links are visited.

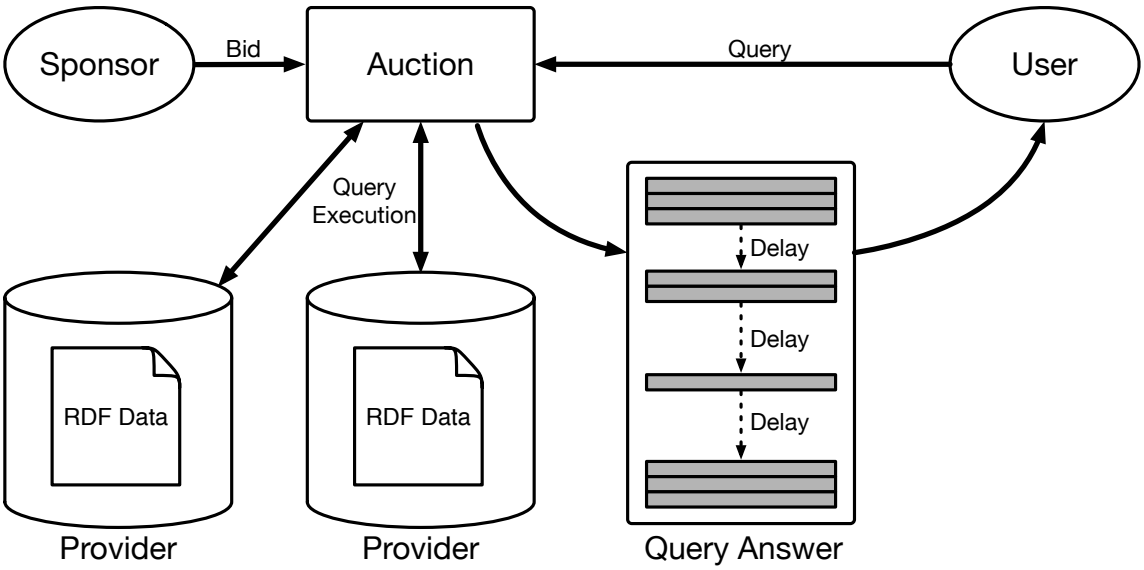


Fig. 4.1: A user gets a delayed query answer based on the bids of sponsors.

To achieve the link redirection functionality, the auction has to replace each occurrence of a sponsored link with a new link that directs the user to the auction when visited. The auction has to track which link replacement corresponds to which original link and redirect the user (or program), accordingly. The revenue generated by the auction is used to finance the provider of the data. Not only the provider providing the sponsored link but all providers providing any relevant data for the solution which contained the visited link gets a share of the revenue. In addition, it is possible to use part of the revenue to subsidize providers which were not so lucky and could not (or not enough) contribute to different sponsored links to cover the operation costs of their services. The exact distribution of the revenue among the providers is beyond the scope of this paper, however. It is important to note that for the auction to work properly, the providers and the auction need to build a closed system, meaning that the providers' data are only accessible through the auction itself and cannot be accessed directly by the user. If the providers and the auction would not build a closed system, a user could circumvent the auction mechanism and directly query the provider's data without suffering from the delays introduced by the auction.

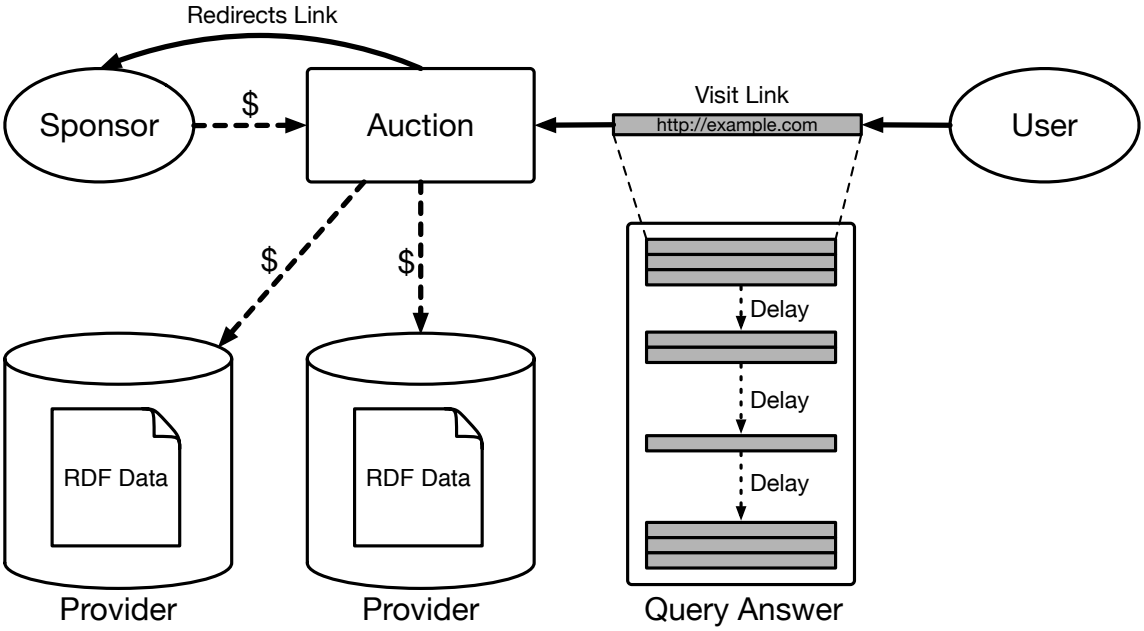


Fig. 4.2: If a user visits a link, the sponsor of that link pays the auction which distributes the money among the providers.

Note that delaying certain solutions is not part of the standard SPARQL protocol [Feigenbaum et al., 2013]. Instead, the query answer must be offered as a *stream* of solutions. There are various extensions of SPARQL which offer the possibility to return solutions in form of a stream [Anicic et al., 2011, Barbieri et al., 2010, Calbimonte et al., 2010, Dell'Aglio et al., 2017, Le-Phuoc et al., 2011, Özçep et al., 2014, Rinne et al., 2012]. A detailed discussion of the advantages or disadvantages of the different streaming solutions is beyond the scope of this paper, however.

4.5 Formal model

In this section, we introduce the formal model of our concept. First, we introduce our new link selection model and then, we discuss the how this model can be applied to a weighted VCG auction.

4.5.1 Batch Link Model

Our *Batch Link Model* differs from traditional models for sponsored search results. In click models for sponsored search results an ad is shown to a user if a bidder bids on a certain *keyword*. This relevance of different ads for a specific keyword can differ quite a lot because (1) the keyword entered by a user can be ambiguous or unspecific because of the lack of any semantics and (2) an advertiser can decide to bid for a certain keyword, even if the ad is not very relevant. In a semantic search, the ambiguity is mostly removed and, given

that the user, or a program acting on behalf of the user, formulated the query diligently enough, the lack of specificity as well. In addition, a bidder cannot force a certain service link to be part of the query answer by increasing the bid enough. Instead, only links which actually match the query are allowed to be in the query answer. Hence, in our model, we assume that each service link i contained in a solution for a query answer has the *same* probability p_{rel} of being relevant for the query. The probability p_{rel} depends on the query the user issued and has to be estimated by the auction.

The model assumes that a user will select and visit only service links which are considered relevant for the query. In addition, a user will visit at most one link. The motivation for this assumption is that, in contrast to advertisement in search result pages, a SPARQL query returns the user exactly those links which are of interest for the user, due to the nature of the semantic search. The user can judge the relevancy of the service link by the information embedded in the solution which contains the link. This is different from a WWW search, where users often have to follow a link to discover whether the content provided by the webpage is actually relevant.

If more than one service link is delivered, the user has to decide which of the relevant links to select. Assume that there are n different links, each having a probability of p_{rel} of being relevant to the user. The probability that service link i is the only relevant link is:

$$p_{\text{rel}} \cdot (1 - p_{\text{rel}})^{n-1} \tag{1}$$

In this case, the user will visit this one link because it is to only one relevant.

However, there might be more than one relevant link. In case there are two relevant service links, each relevant link has the same probability of being visited. Hence, the probability the user will select service link i is half the probability that one of the two relevant links is selected:

$$\frac{(p_{\text{rel}})^2 \cdot (1 - p_{\text{rel}})^{n-2}}{2} \tag{2}$$

In total, there are $(n-1)$ cases having two relevant service links where one of them is link i. In general, there are $\binom{n-1}{k-1}$ cases having k relevant service links where one of them is link i. If we combine all possible cases, we get the formula for the probability $p_{\text{sel}}(n)$ of a link i being selected:

Definition 1. *Given n links with a probability of p_{rel} of being relevant to the user, we define $p_{\text{sel}}(n)$ as*

$$p_{\text{sel}}(n) := \sum_{j=1}^{n} \binom{n-1}{j-1} \cdot \frac{(p_{\text{rel}})^j \cdot (1 - p_{\text{rel}})^{n-j}}{j} \tag{3}$$

Corollary 1.

$$p_{\text{sel}}(n) = \frac{1 - (1 - p_{\text{rel}})^n}{n} \tag{4}$$

Proof.

$$
\begin{aligned}
&\sum_{j=1}^{n} \binom{n-1}{j-1} \cdot \frac{(p_{\text{rel}})^j \cdot (1-p_{\text{rel}})^{n-j}}{j} \\
&= \sum_{j=1}^{n} \binom{n-1}{j-1} \cdot \frac{n}{j} \cdot \frac{(p_{\text{rel}})^j \cdot (1-p_{\text{rel}})^{n-j}}{n} \\
&= \frac{\sum_{j=1}^{n} \binom{n}{j} \cdot (p_{\text{rel}})^j \cdot (1-p_{\text{rel}})^{n-j}}{n} \\
&= \frac{\left(\sum_{j=0}^{n} \binom{n}{j} \cdot (p_{\text{rel}})^j \cdot (1-p_{\text{rel}})^{n-j}\right) - (1-p_{\text{rel}})^n}{n} \\
&= \frac{1-(1-p_{\text{rel}})^n}{n}
\end{aligned} \tag{5}
$$

where the last equality holds because the sum in brackets is the distribution formula of the binomial distribution. □

Introducing delays into the delivery of results adds an additional layer of complexity to our batch link model. Figure 4.3 shows an example of how the probability can decrease over time. In this example, the first batch is delivered in slightly less than 3 seconds and the second one after almost 7 seconds. The probability that the user waits until the first batch of solutions arrives is around 55%. The probability that the user waits until the second batch of solutions also arrives is around 25%. The probabilities Δp_1 and Δp_2 indicate how likely it is that the user stops waiting after receiving batch 1 and batch 2, respectively, and chooses one of the relevant links received. Let n_j be the number of solutions contained in batch j. We can now define the probability of a link inside a specific batch being selected:

Definition 2. *Let there be m batches containing $n_1, \ldots, n_m$ query solutions, each batch j having a probability of Δp_j of being the last one received by the user. A link i contained in batch $b(i)$ has the following probability p_{sel_i} of being selected by the user:*

$$
p_{\text{sel}_i} := \sum_{j=b(i)}^{m} \Delta p_j \cdot p_{\text{sel}}(N_j) \ , \tag{6}
$$

where $N_j = \sum_{k=1}^{j} n_k$

We also define a probability Δp_0 which indicates the likelihood that the user does not wait for the first solution to arrive.

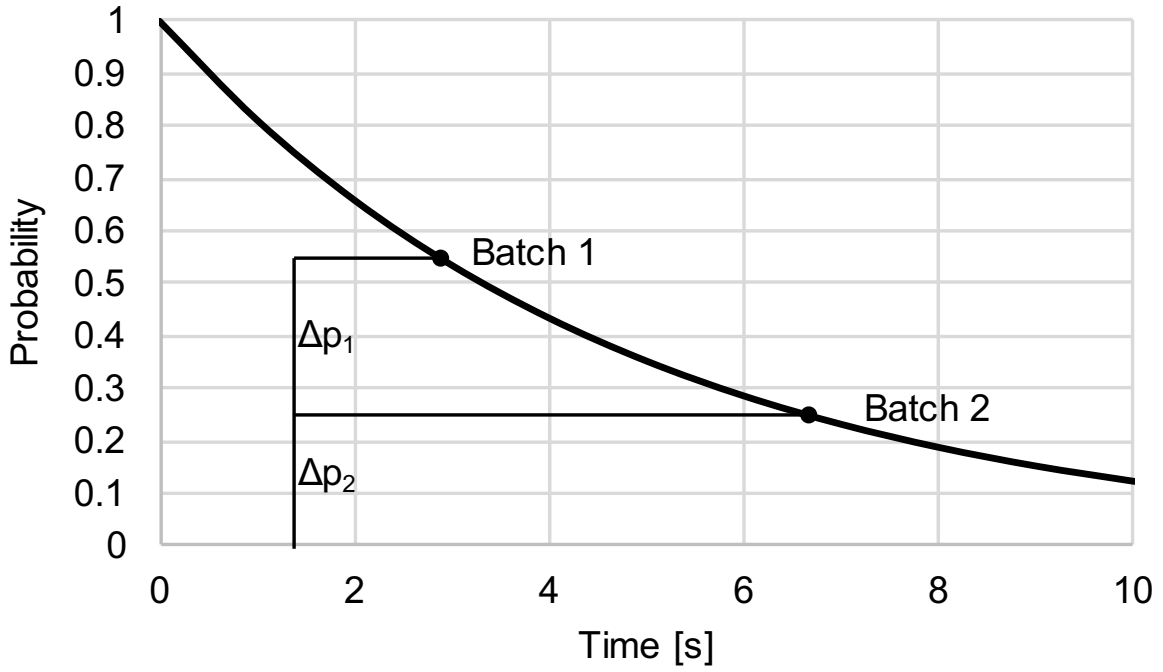


Fig. 4.3: Probability that a user waits for the solutions to be delivered.

4.5.2 Ranking Function

To decide which solution should be placed in which batch, they have to be ranked according to the bids placed on the links. There are two well-known ways of ranking the links, either by ranking them by the bids placed on them or by their *revenue*, which is the probability of being selected times the bid. Because of our assumption that the different links have the same probability of being relevant for the user, two different links will have the same probability of being selected for the same position in the ranking. In addition, the probability of being selected is monotonically decreasing with respect to the position. This means that ranking the links by bids results in the same ranking as ranking them by revenue. Hence, we can just order the solutions by the bids they contain to achieve a ranking by revenue. If two links receive the exact same bid, the ranking among the solutions is determined randomly. Note that using a ranking function is not compatible with a user specified order of the solutions and hence, ORDER-BY clauses in SPARQL queries are not supported by our model.

After the solutions are ranked according to the bids on the links they contain, the solutions can be assigned to the different batches. Let n_j the amount of solutions contained in batch j. The first n_1 solutions will be assigned to the first batch, the next n_2 solutions to the second batch, and so on. The parameters $n_1, \ldots, n_m$ as well as the delay of the different batches have to be determined in advance. The size of the different batches and the delays determine the probability for the different service links of being selected.

4.5.3 Weighted VCG

We use the weighted VCG auction [Nisan and Ronen, 2007] to determine the prices the different bidders must pay for each visit by a user. We use this auction mechanism because it is *truthful*, which means that the best strategy for each bidder in this auction is to place a bid which equals their *value* for a visit by the user. The bidder's value for a visit is the *maximal* amount the bidder would be willing to pay for a user's visit. Since the auction is truthful, we will assume that the bidders are bidding their true value and we will use the two expressions *bid* and *value* interchangeably.

The probabilities which are given by the batches, the delays, and the probability p_{rel} are needed to calculate the prices the sponsors have to pay if the user visits a service link. The VCG price π_i for a link i is based on the "harm" the bidder imposes on all other bidders. This harm can be calculated by the difference of the revenue all other bidders would have if there were no bid on link i minus the revenue all other bidders have because there is a bid on link i. The revenue of bidder i must be equal to this difference.

Definition 3. *Given n links with probabilities $p_{\text{sel}_1}, \ldots, p_{\text{sel}_n}$ and bids $v_1, \ldots, v_n$. The weighted VCG price π_i for link i is:*

$$\pi_i := \frac{\sum\limits_{\substack{j=1 \\ j \neq i}}^{n} v_j \cdot \left(p_{\text{sel}_j}^{\not\ni i} - p_{\text{sel}_j} \right)}{p_{\text{sel}_i}} \tag{7}$$

where $p_{\text{sel}_j}^{\not\ni i}$ denotes the probability of link j being visited if there were no bid on link i.

We show now how this price can be computed more efficiently. For this, we need the following definition:

Definition 4. *For a link j in batch $b(j)$, we define the 2nd value v_j^{2nd} as the highest bid of the next batch $b(j)+1$. If j is in the last batch $b_{\max}$, we define the 2nd value to be 0:*

$$v_j^{\text{2nd}} := \begin{cases} \max\{v_k : b(k) = b(j)+1\}, & \text{for } j < b_{\max} \\ 0, & \text{for } j = b_{\max} \end{cases} \tag{8}$$

Intuitively, the 2nd value v_i^{2nd} is a lower bound for a bid v for staying in batch $b(i)$. As long as $v > v_i^{\text{2nd}}$, the link can stay in batch $b(i)$. If $v = v_i^{\text{2nd}}$, it is not guaranteed that link i stays in the same batch because the ranking among all links with the same value is random. Finally, if $v < v_i^{\text{2nd}}$, the link ends up in a next higher batch.

Using this definition, we can state the following theorem:

Theorem 1. *The weighted VCG price π_i for a link i is given by:*

$$\pi_i = \frac{\sum\limits_{j=b(i)}^{m} v_j^{\text{2nd}} \cdot \Delta p_j \cdot p_{\text{sel}}(N_j)}{p_{\text{sel}_i}} \tag{9}$$

where m is the total number of batches.

Proof. Equation 7 gives the general form of the weighted VCG price. The first observation is that $p^{\not\ni i}_{\mathrm{sel}_j} = p_{\mathrm{sel}_j}$ for any link j which is positioned before link i, as their position does not change when the bid on link i is removed. For any link j positioned after link i, the probability $p^{\not\ni i}_{\mathrm{sel}_j}$ changes only if link j ends up in a different batch because of the removal of the bid on link i.

There might be multiple service links which have a zero bid on them. Any change in the rankings among links without any bid yields $v_j \cdot \left(p^{\not\ni i}_{\mathrm{sel}_j} - p_{\mathrm{sel}_j}\right) = 0$ because $v_j = 0$ and hence, it does not affect the price. Hence, without loss of generality, we can assume that link i gets the last possible position in the ranking when the bid is removed.

If link i receives the last position, the link with the highest value in each batch between $b(i)$ and $b_{\max}$ ends up in a next lower batch. This means that $v_j \cdot \left(p^{\not\ni i}_{\mathrm{sel}_j} - p_{\mathrm{sel}_j}\right)$ can only be non-zero if $v_j = v^{\mathrm{2nd}}_{b(j)-1}$ that is, if v_j is the highest value inside its batch $b(j)$ and hence, the second value of the next higher batch. In these cases, the difference $p^{\not\ni i}_{\mathrm{sel}_j}$ and p_{sel_j} is exactly:

$$\Delta p_j \cdot p_{\mathrm{sel}}(N_j) \tag{10}$$

by Equation 6. Hence:

$$\sum_{\substack{j=1 \\ l \neq i}}^{n} v_j \cdot \left(p^{\not\ni i}_{\mathrm{sel}_j} - p_{\mathrm{sel}_j}\right) = \sum_{j=b(i)}^{m} v_j^{\mathrm{2nd}} \cdot \Delta p_j \cdot p_{\mathrm{sel}}(N_j) \tag{11}$$

□

4.6 Optimizing Batch Sizes and Delays

There are two sets of parameters we must control when setting up the delay auction: the batch sizes $n_1, \ldots, n_m$ and their probabilities $\Delta p_1, \ldots, \Delta p_m$.

Remember that we have a probability Δp_0 indicating how likely it is that the user does not wait for the first batch of solutions to arrive. The probability Δp_0 is given by the user model and cannot be set by the auction designer. In addition, we set a threshold $t_{\max}$ which indicates the maximal amount of time we are willing to let the user wait until the last solution arrives. The threshold $t_{\max}$ maps to a probability $p_{\max}$ that the user waits for the last batch to arrive.

Let there be n different links. We have three constraints:

$$\sum_{i=1}^{m} n_i = n \ , \tag{12}$$

$$\sum_{i=1}^{m} \Delta p_i = 1 - \Delta p_0 \ , \tag{13}$$

and

$$\Delta p_m \geq p_{\max} \ . \tag{14}$$

After the probabilities $\Delta p_1, \ldots, \Delta p_m$ are determined, the delays of the batches can be chosen to match the desired probabilities.

Note that the parameters $n_1, \ldots, n_m$ and $\Delta p_1, \ldots, \Delta p_m$ cannot be decided ad hoc when the bids are received. If one would try to optimize the parameters given the bids one would render the auction untruthful despite the use of the weighted VCG mechanism. This is because before the mechanism is applied, the bidders could manipulate the parameters of the auction by manipulating their bids. Hence, the optimizations we discuss in this section have to be done based on what values have to be expected coming up in future auctions.

4.6.1 Optimizing Revenue

The revenue r generated by our auction mechanism is the sum of the VCG payments of all sponsors involved in a specific query answer. As we showed in Section 4.5.3, the payments can be calculated according to Equation 7. The total revenue generated by the auction is:

$$\begin{aligned} r &= \sum_{i=1}^{n} \pi_i \cdot p_{\text{sel}_i} \\ &= \sum_{i=1}^{n} \sum_{j=b(i)}^{n} v_j^{\text{2nd}} \cdot \Delta p_j \cdot p_{\text{sel}}(N_j) \end{aligned} \tag{15}$$

If we concentrate first on the best choice for the probabilities $\Delta p_1, \ldots, \Delta p_m$, we see that the revenue is a linear function with respect to those probabilities. This means that the revenue is maximized when one of the probabilities is set to 1 and all others to 0. Let Δp_{max} be the probability which maximizes the revenue when set to 1. If we also consider the constraint in Equations 13 and 14, the revenue is maximized when $\Delta p_{max} = 1 - \Delta p_0$ and $\Delta p_m = p_{\max}$ or, if $max = m$, $\Delta p_{max} = 1 - \Delta p_0$, respectively. All other probabilities are set to 0. Setting $\Delta p_j = 0$ means that batch j has the same delay as the next batch $j + 1$ and practically, they become one batch.

Optimizing the batch sizes is not as straightforward as optimizing the probabilities. But, since there are at most two batches remaining after optimizing the probabilities, the problem becomes easier. If $max = m$ the revenue is 0, as there is only one batch and hence, the second value is 0. If there are two batches left we redefine the two remaining probabilities as Δp_1 and Δp_2 and denote with n_1 and n_2 the number of solutions in the two batches. We will denote the revenue when having only two batches as r^*:

$$\begin{aligned} r^* &= \sum_{j=1}^{n_1} v_1^{\text{2nd}} \cdot \Delta p_1 \cdot \frac{1 - (1 - p_{\text{rel}})^{n_1}}{n_1} \\ &= v_1^{\text{2nd}} \cdot (1 - \Delta p_0 - p_{\max}) \cdot (1 - (1 - p_{\text{rel}})^{n_1}) \end{aligned} \tag{16}$$

Note that the value v_1^{2nd} depends on the choice of n_1. The only way to optimize r^* is to iterate through all possible values of n_1. Fortunately, the possible values for n_1 are discrete and bounded by n and hence, the problem can be solved in $O(n)$ time.

4.6.2 Optimizing Social Welfare

The social welfare s generated by our auction mechanism is the sum of the values of all links times the probability that the link is visited. Since the auction is truthful, we can assume that the bids equal the bidders actual value for being selected. Hence, the social welfare is given by the following formula:

$$\begin{aligned} s &= \sum_{i=1}^{n} v_i \cdot p_{\text{sel}_i} \\ &= \sum_{i=1}^{n} v_i \cdot \left(\sum_{j=b(i)}^{m} \Delta p_j \cdot p_{\text{sel}}(N_j) \right) \end{aligned} \tag{17}$$

The same argument which applies to maximizing the revenue also applies here: the social welfare is a linear function with respect to those probabilities, which is maximized if we set $\Delta p_{max} = 1 - \Delta p_0 - p_{\max}$ and $\Delta p_m = p_{\max}$, or $\Delta p_{max} = 1 - \Delta p_0$, respectively, and all others to 0. Again, to optimize social welfare the auctioneer has to split the solutions into at most two batches. Similar to r^*, we denote the social welfare when having two batches as s^*:

$$\begin{aligned} s^* = &\left(\sum_{i=1}^{n_1} v_i \right) \cdot (1 - \Delta p_0 - p_{\max}) \cdot \frac{1 - (1 - p_{\text{rel}})^{n_1}}{n_1} + \\ &\left(\sum_{i=1}^{n} v_i \right) \cdot p_{\max} \cdot \frac{1 - (1 - p_{\text{rel}})^{n}}{n} \end{aligned} \tag{18}$$

As with r^*, s^* can be maximized by iterating through all possible values for n_1. There are two properties of the maximal value for s^* worth noting. First, the parameter n_1 which maximizes s^* does not maximize r^* in general. This is easy to see when comparing Equation 16 with Equation 18: r^* depends on the value v_1^{2nd} whereas s^* depends on all values. Second, choosing $n_1 = n$ does not maximize s^* in general. This is because the more solutions are included in the first batch, the more likely it is that a solution having a low value will be chosen.

4.6.3 Optimizing User Experience

The user's experience is an important aspect of our auction we have not discussed so far. Even if the data the auction is offering to the user is for free, a user might switch to a competitor offering a similar service if the competitor can offer a better user experience.

We have already seen that in both cases, optimizing revenue and optimizing social welfare, the optimal choice of parameters results in two batches, one delivered as fast as possible and the second one delivered at time $t_{\max}$. Delaying a potentially big chunk of the available solutions for this maximal delay $t_{\max}$ might damp the user's experience, however. Of course, the user would prefer to receive all data in the first batch, which is delivered immediately. However, as we have seen in the previous section, delivering all

data at once results in zero revenue and possibly suboptimal social welfare. Fortunately, it is possible delaying some solutions without impacting the user experience too much. The main argument is that a user will need some time to consume the solutions contained in a batch. If the delay of the next batch is not larger than the time the user needs to consume the current batch, the user will be able to seamlessly consume all the solution from the different batches.

Another aspect which can damp the user's experience is the fact that the delivery of the data is biased in the sense that highly sponsored solutions are always delivered first. In Section 4.7, we discuss an extension of our model which reduces this bias.

4.6.4 Simulation

In this section, we illustrate how revenue and social welfare behave when there are only two batches. Figure 4.4 shows four different value distributions which we labeled with Linear, Tableau, Dominant, and Equal, respectively. For our simulation, we set $\Delta p_0 = 0.05$, $p_{\text{rel}} = 0.6$, and $p_{\max} = 0.1$.

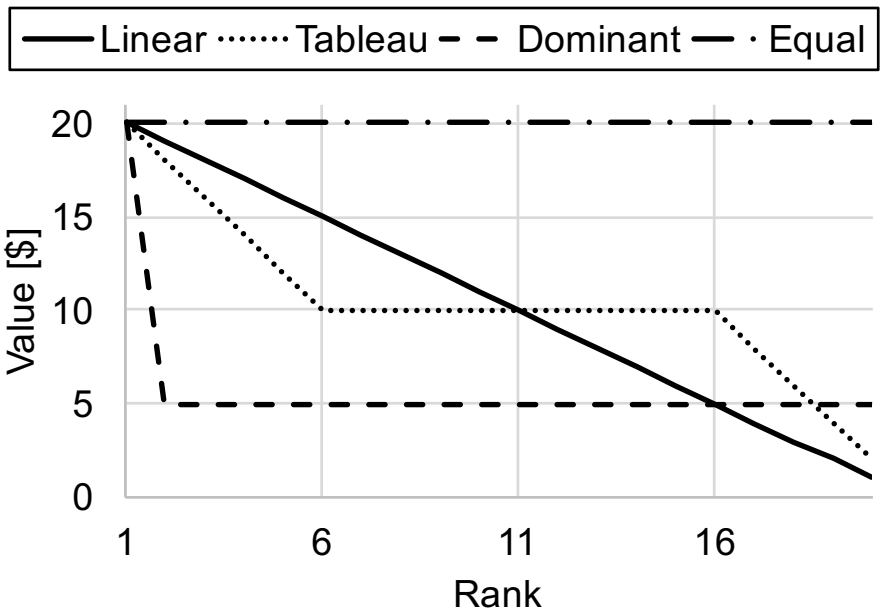


Fig. 4.4: Value distribution for four different scenarios.

Figure 4.5 shows the revenue for different choices of n_1. The revenue always starts at \$0 for $n_1 = 0$ and closes with \$0 for $n_1 = 20$. This is no surprise, as in both cases all solutions are only assigned to a single batch and hence, the second price is \$0. There are two remarkable observations about the revenue, however. First, for the Tableau distribution, the revenue stays the same for $n_1 = 5$ up to $n_1 = 15$. This happens because the second value stays constant within this range, as the bids are all the same. Second, the Dominant

and Equal distributions both have their maximum at $n_1 = 19$. The reason for this is that the highest bid within any distribution does not influence the revenue.

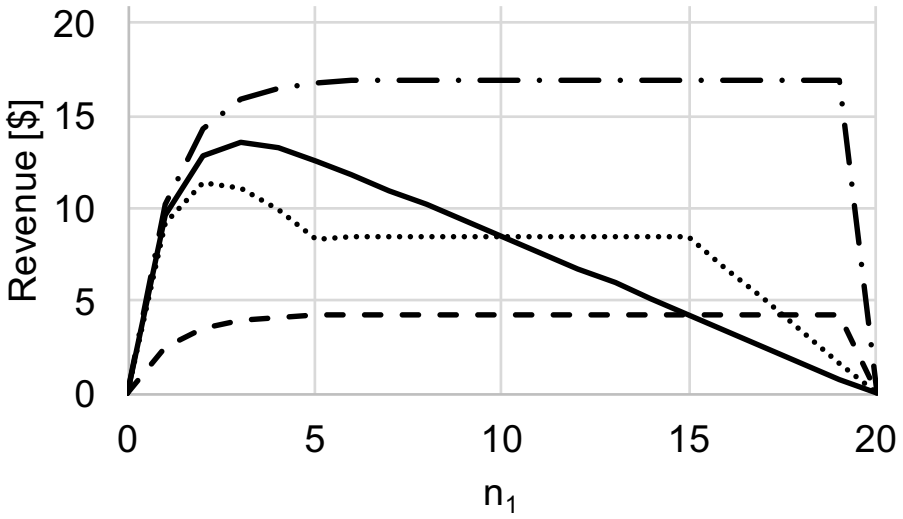


Fig. 4.5: Revenue for different values for n_1 for the different value distributions from Figure 4.4.

Figure 4.6 shows the social welfare for different choices of n_1. The social welfare is always positive, no matter the choice of n_1. For the Equal distribution the social welfare is maximized for $n_1 = 20$, but does not change much after n_1 is beyond seven. The values for n_1 which maximize the social welfare for the Linear and Tableau distributions, respectively, are quite close to the ones which maximize their respective revenue. The Dominant distribution has its maximum for the social welfare for $n_1 = 1$. Increasing n_1 further increases the chance that a link with a lower value is chosen by the user and hence, decreases social welfare. The Dominant distribution illustrates how far away the values for n_1 can be which maximize social welfare and revenue, respectively.

4.7 Extension

To mitigate the bias our auction introduces by delivering the highest sponsored data first, we propose an extension of our original model. The idea of this extension is to auction off only part of the available positions in the batches and assign the rest of them randomly. The result of this extension is similar to what users experience when using search engines: Part of the results is sponsored content and ranked according to some bids, the other part of the results is unbiased and only depends on what the user is searching for.

The first step in our extension is to designate some slots in each batch which should be reserved for sponsored solutions. We will denote with n_i^{spons} and n_i^{rand} how many of the n_i slots in batch i are reserved for sponsored solutions or are assigned randomly, respectively.

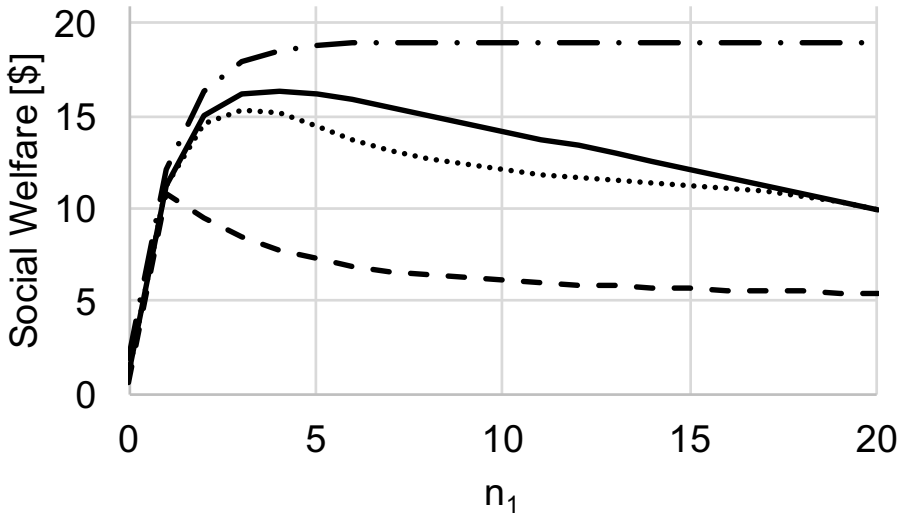


Fig. 4.6: Social welfare for different values for n_1 for the different value distributions from Figure 4.4.

The next step is to assign solutions to the random slots. We denote with n^{spons} and n^{rand} the total number of sponsored slots and random slots, respectively, and with n the total number of slots, which equals the total number of solutions for the query. Note that once we have assigned all random slots, there are $n - n^{\text{rand}}$ solutions which did not receive any slot, yet. These solutions are randomly arranged in a waiting queue, where the first link in the queue will be the first one to occupy a newly open random slot.

The next step is to assign solutions to the sponsored slots. We start with the assignment of the n_1^{spons} slots in the first batch. For this, we select the n_1^{spons} solutions with the highest bid from all those solutions which are not already assigned to the first batch by the random assignment. This means that some of the solutions get upgraded to the first batch. An upgraded solution can be either one of the solutions in the waiting queue or one of the solution which already had a randomly assigned slot. In the latter case, the upgrading creates a new open position for a randomly assigned slot, which is assigned to first link in the waiting queue. This procedure is applied sequentially to the sponsored slots n_2^{spons} to n_m^{spons}. For every batch, we only consider those solutions which are currently in a higher (and later) batch or the waiting queue.

Once we have the new ranking of the solutions, we can again calculate the weighted VCG prices according to Definition 3. Note that the Theorem 1 does not anymore apply to this new setting.

We conclude our extension with an example:

Example 1. Assume that there are four links with bids of \$0, \$10, \$20, and \$30. The left part in Figure 4.7 shows a random assignment of the bids \$0 and \$30. The other two bids, \$10 and \$20, are assigned to the waiting queue. The bid \$20 takes precedence over \$10 inside the queue. The right part in Figure 4.7 shows the final assignment of the bids. The arrow labeled Ⓐ indicates the first step of the assignment of the sponsored slots:

bid \$30 gets to the first sponsored slot, because it is the highest bid of those three bids, \$30, \$20, and \$10, which are not yet in batch 1. Next Ⓑ, the bid \$20 gets the random slot previously occupied by \$30, because bid \$20 has the priority in the queue. Finally Ⓒ, bid \$10 gets the last remaining sponsored slot because it is the highest (and only) bid which did not yet get a slot in batch 1 or 2. The weighted VCG price is \$0 for all solutions which got a randomly assigned slot, eventually, because changing their bid to \$0 does not influence any other assignments of slots. This means that the bidders with bids \$0 and \$20 do not pay if the user visits their link. The bidder with bid \$10 also gets a price of \$0. This, because the bidder would have gotten in the second batch even with a bid of \$0. Finally, the weighted VCG price for the bidder with \$30 is the value all other bidders would have if bid \$30 would have been \$0, minus the value all other bidders have for the current assignment.

Assume that $p_{\text{rel}} = 0.8$, $\Delta p_1 = 0.5$, and $\Delta p_2 = 0.4$. Hence, the probabilities of being selected for the two first slots are 0.20096 each, and for the second two slots 0.04096 each. If bid \$30 would be zero, the first sponsored slot would be assigned to bid \$20 and the first random slot would still belong to the bidder with the original bid \$0. The second sponsored slot would be assigned to bid \$10. The second random slot would be assigned to the bidder with the original bid of \$30. The total value of this assignment is \$4.4288. The value of all other bidders for the original assignment is \$1.2288. With a probability of 0.20096 of being selected, the weighted VCG price for the bidder with bid \$30 is \$15.92, which has to be payed if the user visits the link.

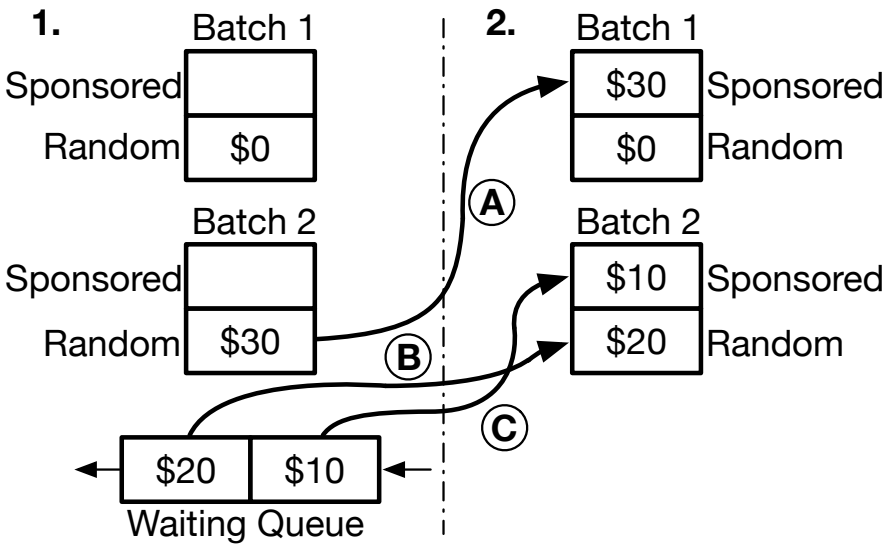


Fig. 4.7: Example: four links get assigned to different slots.

4.8 Limitations and Conclusions

We introduced a new concept of a delayed-anser auction to finance free data in the WoD. As we have seen, the choice of parameters can influence the generated revenue, generated social welfare, and the user experience. In general, it is not possible to find parameters which maximizes all three of them. Hence, it is the choice of the auction designer to find a suitable trade-off between revenue, social welfare, and user experience. We also discussed an extension which reduces the bias we introduce by prioritizing solutions containing links with high bids.

Our new auction model is not restricted to the use case of the WoD. It can be applied to any setting where multiple bidders can occupy the same slot and different slots have a decreasing probability of being selected. An additional restriction is that we assume that all solutions have the same relevance to the user. As we discussed, this assumption is reasonable in the WoD setting. In other settings, this assumption might not apply. In this case, the auction might become more complicated because the probabilities of being selected might not be separable anymore and hence, the weighted VCG auction is not applicable.

One additional assumption we made is that there is only one sponsor per solution. While this assumption might be true in most cases, it is possible that a user might issue a query which contains multiple sponsored links per solution. In such situations, the bid per solution can be defined as the sum of the bids for each link contained in the solution. With this definition, we could still apply our auction model, but we would have to find a way to distribute the weighted VCG payment among the bidders which placed a bid on the same solution. If one decides to distribute the payments proportional to the bids placed on the links, the auction would not anymore be truthful, however.

What is left for future work is the distribution of the generated revenue among the data providers. The revenue can be used to finance those providers which proved to be important for answering queries or subsidize those providers which struggle the most to keep their services running.

With our concept of a delayed-answer auction, we provided a first sponsored auction model for the Web of Data. Whatever the limitations of our concept are, it represents a first model for a financially sustainable and free Web of Data.

Chapter 5

Save Money by Price Sharing for Streaming Data

This chapter is based on:

Tobias Grubenmann, Daniele Dell'Aglio, Abraham Bernstein, Dmitry Moor, and Sven Seuken. 2018. ***Price Sharing for Streaming Data: A Novel Approach for Funding RDF Stream Processing****. Manuscript available at https://2018.eswc-conferences.org/wp-content/uploads/2018/02/ESWC2018_paper_5.pdf*

Price Sharing for Streaming Data: A Novel Approach for Funding RDF Stream Processing

Tobias Grubenmann, Daniele Dell'Aglio, Abraham Bernstein, Dmitry Moor, and Sven Seuken

Department of Informatics, University of Zurich, Switzerland

Abstract. RDF Stream Processing (RSP) has proposed solutions to continuously query streams of RDF data. As a result, it is today possible to create complex networks of RSP engines to process streaming data in a distributed and continuous fashion. Indeed, some approaches even allow to distribute the computation across the web. But both producing high-quality data and providing compute power to process it costs money.
The usual approach to financing data on the Web of Data today is that either some sponsor subsidizes it or the consumers are charged. In the stream setting consumers could exploit synergies and, theoretically, share the access and processing fees, should their needs overlap. But what should be the monetary contribution of each consumer when they have varying valuations of the differing outcomes?
In this article, we propose a model for price sharing in the RDF Stream Processing setting. Based on the consumers' outcome valuations and the pricing of the raw data streams, our algorithm computes utility-maximizing prices different consumers should contribute whilst ensuring that all the participants have no incentive of manipulating the system by providing misinformation about their value, budget, or requested data stream. We show that our algorithm is able to calculate such prices in a reasonable amount of time for up to one thousand simultaneous queries.

5.1 Introduction

The task of processing RDF data continuously in the form of streams has spawned various extensions to SPARQL query processing systems [Anicic et al., 2011, Barbieri et al., 2010, Calbimonte et al., 2010, Le-Phuoc et al., 2011]. RDF streams are produced and consumed continuously, as opposed to static RDF data. The easy integration of RDF data allows one to join, merge, aggregate, filter, and sample such data from various different sources and to produce new RDF streams. In contrast to traditional stream processing, RDF Stream Processing (RSP) over the Web of Data (WoD) has two important unique features: (1) RSP can be distributed over various loosely connected machines on the web [Mauri et al., 2016, Taelman et al., 2016] and (2) different RDF queries have a much higher chance of overlapping computation because of the use of global identifiers. This unique combination of distribution and overlap offers new opportunities of exploiting synergies by sharing resources to reduce the overall workload. At the same time, processing streams is a continuous activity that may require a high amount of resources—both source data and computing power—on an ongoing basis.

To date, research in the WoD has completely overlooked the question of how these resources should be paid for [Grubenmann et al., 2017b]. Today, in most Web of Data applications, processing is being paid for by a single entity that either profits from the data being processed in some way or has decided to subsidize it (e.g., to facilitate research or economic growth). In the streaming domain, these processing fees would be incurred

continuously as long as the streamed data is consumed. *Also, given the distributed and potentially overlapping nature of RSP as well as possibly varying valuation for the outcomes, it is unclear how to share the processing fees.* Pricing schemes for Cloud Computing do not apply to this setting, unfortunately. This is because in cloud computation, the computational resources are split up between the customer and each of them pays an appropriate price share. In our setting, however, the output of a single process can be shared among different entities. This adds an additional layer of complexity which is not considered in pricing schemes for cloud computation.

As an example, consider the processing of Live TV viewership information. Channels might be interested in audience demographics, advertisers in how many people have seen an advertisement for a specific product, and recommender systems are interested in viewing histories. Each of the three entities: channel providers, advertisers, and recommender systems have their own streaming query. However, there might be some overlap in the computation they perform on these data and hence, some output can be reused for several different queries. As a result, computational resources can be saved by doing the respective computation only once and, in turn, the channel providers, advertisers, and recommender systems can save some money by sharing the price of this computation.

How much should each of them contribute to the payment for gathering and processing the various pieces of data that answer their differing but overlapping queries?

This paper proposes a novel model to study the sustainability of continuously processing RDF data. It takes the joint execution plan, which is based on the collected queries of all data consumers, of existing multi-query optimizers as an input. Then, it combines this query execution plan with the outcome valuations and run-time limits (i.e., the length of time during which a query answer is needed) from all data consumers, the pricing of the raw data streams, and the pricing of the computation to determine a utility-maximizing payment distribution (i.e., it ensures that each consumer gets the most value out of the price they pay). Assuming that the prices of the raw data are given, our payment allocation ensures that the data consumers have no incentive of manipulating the system by providing misinformation about their value, budget, or requested data stream—a property economists call truthfulness.

Our contributions are, hence, as follows: (1) we introduce a formal setting for price sharing in RDF Stream Processing (which to our knowledge is the first attempt to do so), (2) we present a set of requirements that a price sharing system should fulfil, (3) we propose a model for price sharing that maximizes the data consumer's benefits (or utility) whilst incentivizing them to be honest about their desiderata, and (4) evaluate the runtime and the savings of money for our proposed model empirically showing that our algorithm is able to calculate such prices in a reasonable amount of time for up to one thousand simultaneous queries.

The paper is structured as follows. First, we will discuss some related work and introduce some motivating examples which would benefit from our market model. Next, we introduce a formal model, show how to solve the cost-sharing problem, and evaluate the runtime and cost savings of the overall system. Finally, we discuss some limitations of our work and conclude our paper.

5.2 Related Work

RDF Stream Processing: The origins of RDF Stream Processing [Dell'Aglio et al., 2017] can be found on the combination of Semantic Web technologies with Information Flow Processing (IFP) systems [Cugola and Margara, 2012]. IFP solutions can broadly be grouped in Data Stream Management (DSMS) systems [Babock et al., 2002] and Complex Event Processing (CEP) systems [Luckham, 2001].

The former ones typically use operators (e.g. sliding windows) to convert portions of the streams in relations, to be managed with classical relational algebra operators, whilst the latter ones analyze the stream to detect relevant event patterns (e.g., sequences).

The two paradigms share the idea of continuous query answering, i.e., the query is issued once and results are computed at several time instants to take into account the availability of new data in the stream.

DSMS inspired several languages such as C-SPARQL [Barbieri et al., 2010], CQELS-QL [Le-Phuoc et al., 2011] and SPARQL$_{stream}$ [Calbimonte et al., 2010], that extend SPARQL to achieve continuous query answering on RDF streams. Similarly, the CEP paradigm inspired languages as EP-SPARQL [Anicic et al., 2011] and STARQL [Özçep et al., 2014]. Finally, INSTANS [Rinne et al., 2012] follows a different approach: instead of extending SPARQL, it performs continuous evaluation of interconnected SPARQL queries.

These new solutions allowed for new and interesting applications. [Wagner et al., 2010] describes a scenario where information from energy producers, grid operators, and appliance manufacturers must be collected and processed. Smart city projects [Puiu et al., 2016, Tallevi-Diotallevi et al., 2013] aim at processing stream data produced by the daily city life to optimize the city's operational tasks.

Financial Cost Sharing: Shapley defined in [Shapley, 1953] two key axioms for cost sharing: (1) the cost shares should be additive dependent on the total costs and (2) if a participant demand does not increase the overall cost, the cost share should be zero. [Aumann and Shapley, 1974] introduced the Aumann-Shapley rule, which charges each participant prices based on the integral of the marginal cost. This pricing rule was successfully applied by [Billera et al., 1978] to a telephone billing problem.

Serial Cost Sharing [Moulin and Shenker, 1992] is a pricing method which is robust to coalition deviations if participants are not allowed to redistribute their outputs. More recently, [Moulin and Laigret, 2011] introduced a method of dividing up the costs of a network when different participants require different subsets of the resources. Our method expands the equal need cost share defined in [Moulin and Laigret, 2011].

None of the related work addressed cost sharing in a setting where different participants require the resources for different, overlapping time periods and participants can potentially manipulate the model by misinformation.

5.3 Motivating Example

Consider a setting where different sources for streaming RDF data provide viewership data for TV channels, the Electronic Program Guide (EPG) data for those channels, activity data from social networks, and news feeds from news networks. Here, there are various

consumers who are interested in these data sources: TV channels can use them to learn which TV shows are the most popular and improve their programming, advertisement networks could use them to improve their advertising strategies based on the people's current interests, and providers of recommender services can use them to improve their recommendations.

Since all these consumers are interested in the same underlying sources, there can be some opportunity for price sharing. Figure 5.1 shows a possible scenario, where a TV channel, an ad network, and a recommender system are interested in the same data but require different computation on these data. The arrows in the figure indicate the flow of data. In our example, the TV channel is only interested in the combination of viewership data and the EPG data, which results in data about the viewers of a specific show. The streaming query of the TV channel accesses this viewer-per-show data and processes it further. The ad network and the recommender system are also interested in combining the viewer-per-show data with some data from social networks to include to social media responses. Each of the two, the ad network and the recommender system, does some additional computation on the data specific for their individual requirement. As obvious from the figure, the price of some computational nodes and some of the sources could be shared by the consumers.

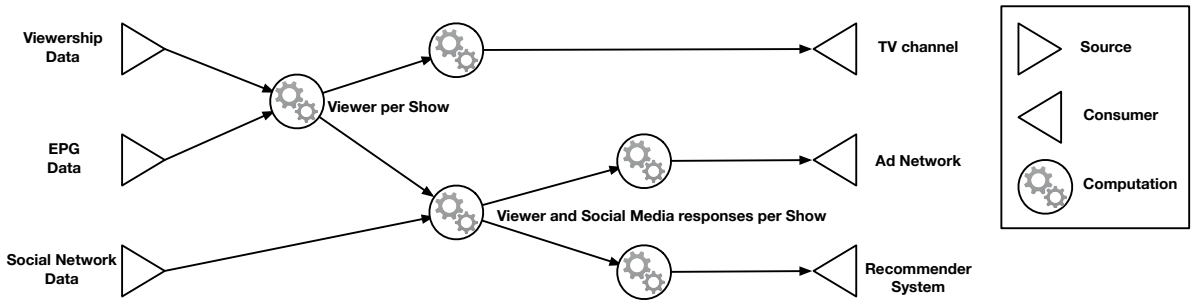


Fig. 5.1: TV channels, ad networks, and recommender systems are interested in the same data but require different computation.

5.4 Model

We build our model for price sharing in stream processing based upon the general architecture of stream processing [Carney et al., 2002, Cugola and Margara, 2012, Margara et al., 2014, Stonebraker et al., 2005], which means that our model can be applied to any instance of this general architecture, irrespective of the exact details of the implementation. In this general architecture, *sources* produce streams of data which are processed by operators producing new streams, which are eventually propagated to interested actors. A *consumer* can submit a *query*, which captures what computation should be performed. As the operators are processing data continuously over time, one can associate a price per

time for each operator. Likewise, a price per time can be associated to each source that streams data into our model. The price per time for sources and operators is determined by the provider of the respective service.

With the sharing of the price of sources and operators comes the question of how these fees should be distributed to the different consumers. For this, our model needs some additional information in addition to the queries and prices. Precisely, our model needs to know from the consumer their *value coefficient*, *budget*, and *time limit* for the query.

Value and value coefficient: The value coefficient indicates how much a consumer is maximally willing to pay for running the query for a certain time. As with the price, the value is given as a constant which indicates the value per time. Multiplying the value coefficient with the actual runtime gives the consumer's *value* for running the query during the allocated runtime.

Budget: The budget indicates how much the consumer is willing to pay in total for running the query. The consumer's payment has to be lower or equal the budget.

Time Limit: The time limit indicates how long the consumer is interested in processing the streaming data, at most. Similar to the budget, the time limit acts as an upper bound but on the runtime of the query instead of the payment. We use a time limit instead of a fixed time for the stream query execution because the latter might violate the value or budget constraints of the consumer.

The combination of value, budget, and time limit allows a consumer some control over which allocations of run-time and prices would be beneficial for him and which allocations would not. Our model maximizes the consumer's *utility*, which is the value for the allocated runtime minus the payment she/he has to pay.

In the following sections, we first define the requirements we want our model to fulfil, followed by a discussion about how to distribute the fees. Next, we discuss how total prices and runtimes of queries can be calculated based on these distributed fees. Afterwards, we discuss to which degree our model meets our requirements.

5.4.1 Requirements for a Sustainable Model

In this section, we identify three requirements that a price sharing model for stream processing should satisfy in order to ensure a sustainable model.

Benefits from participating: It is key to ensure that each consumer benefits from its participation in our model. Each consumer should have as least as high utility as he would have by running his query in isolation, without collaborating and sharing prices with others. In other words, the benefit should consist of an increment in utility, whenever this is possible due to overlapping sources or operators. This leads to our first requirement:

[R1] A consumer must have strictly higher utility when there is some overlap of sources or computation with other queries.

This requirement is crucial to ensure that consumers gain advantage for using our model. Note that Requirement R1 assumes that the customer has strictly higher utility than running the query outside of our model, if there is any potential for price sharing. In

the case that a query has no overlapping computation or sources with other queries, there is no opportunity for price sharing and the customer should have at least as high utility as when not participating in our model. If the model fails to achieve this requirement, it would not be very attractive for consumers.

No benefits from misreporting paramters: The model needs the following parameters as input from the user: value coefficient, budget, and time limit. As these parameters influence the price sharing mechanism, we have to investigate how participants could benefit by misreporting these parameters. Misreporting in this context means reporting a parameter which deviates from the participants true parameter in order to gain an advantage. Requirement R2 makes the model robust against manipulations from the customer with respect to value, execution time limit, and budget:

[R2] No consumer must be able to benefit by misreporting value coefficient, time limit, or budget.

If the model would encourage such manipulations, directly or indirectly, it would enable some consumers to increase their utility by gaming the model. The problem is that such gains in utility are always at the expense of other consumer's utilities. This is because our model maximizes each consumer's utility given the utilities of the other participants and hence, any further gain in utility must result in a decrease of utility for some other participant.

No benefits from query deviation: The last requirement ensures that there is no benefit for a consumer to strategically change his original query to increase utility.

[R3] No consumer must be able to benefit from deviating from his original query.

This means that a customer should not be able to get a higher utility by changing his or her query. Again, if this were the case, a consumer could increase his utility at the expense of other consumers' utilities.

5.4.2 Price Sharing

The algorithm for the price sharing extends the *equal need* cost sharing method presented in [Moulin and Laigret, 2011] by introducing value coefficient, budget, and time limit as additional paramters. In *equal need* cost sharing, each participant receives an equal share of the costs. Applying this to our setting, each consumer who requires a source/operator for his computation receives an equal share of the price of this source/operator. The total price for a consumer's query is the sum of all the price shares of all required sources and operators.

This algorithm has a time complexity of $O(n)$, where n is the number of queries. This is because for each query one has to determine the relevant sources and operators. This number is determined by the kind of query and does not grow with the number of queries. After the relevant sources and operators are determined for each query, the costs of each source and operator have to be distributed. This has a time complexity of $O(s+o)$, where s is the number of sources and o is the number of operators. In the worst case, each query

introduces a certain number of new sources and operators which do not coincide with existing sources and operators. Hence, both s and o grow in the worst case linearly with n. In the best case, all relevant sources and operators for a new query are already part of the topology of sources and operators and s and o do not increase. Therefore, the time complexity is $O(n)$. Figure 5.2 shows an example of how the prices are distributed. The framed labels indicate all the queries for which the source/operator is relevant and the associated price for each query.

Note that different queries can have different runtimes in our model. Hence, the cost of a query varies over time. The algorithm calculates the cost distribution for a given topology of sources and operators. As soon as a new query arrives in the model or a running query stops execution, the topology changes and the price distribution is recalculated.

A function `calculatePriceShares` embeds the algorithm for distributing the prices. We will refer to this function in the next section.

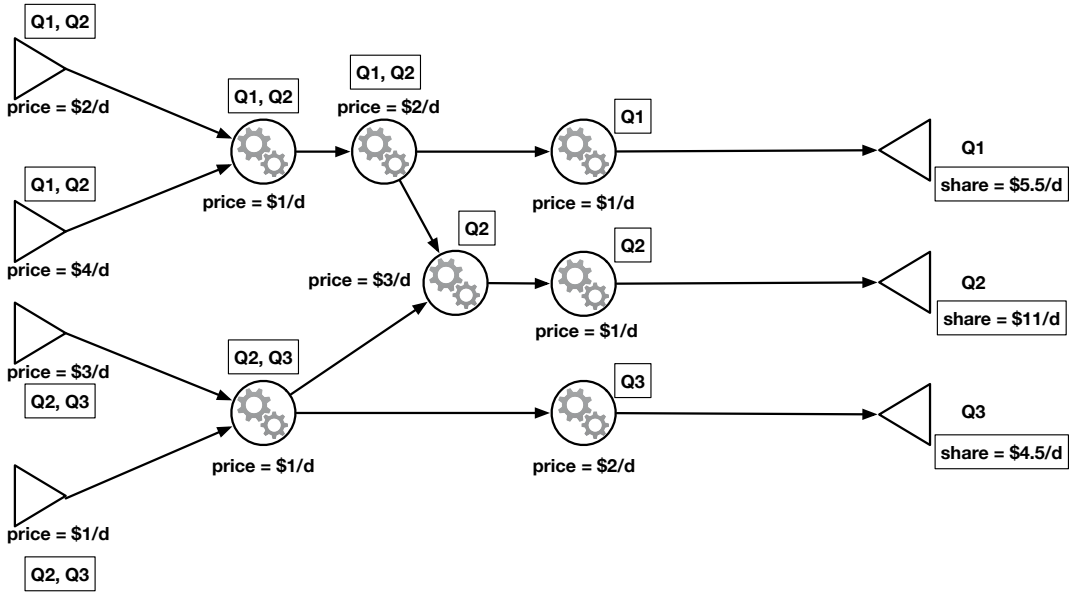


Fig. 5.2: Distribution of prices among different queries

5.4.3 Allocation and Pricing

After the calculation of the associated price for each query, our model can determine (1) how long each query should be executed, which we call the *allocated runtime* and (2) how much this query execution should cost for the consumer, which we call the *total price*.

Algorithm 5.1 shows how runtime and payment are computed given the value coefficients, budgets, and time limits of the consumers. After the return values are initialized (Lines 20 and 20), a set S of indices ranging from 1 to n is initialized (Line 20). Each index $i \in S$ relates to a specific query q_i. The set represents all queries for which the computation of prices and runtime is still in process.

The algorithm calculates prices and runtimes in a loop until this set is empty (Lines 20 to 30). For this, the algorithm enters in another loop (Lines 21 to 25) which calculates the price share p_i for each query q_i for $i \in S$ according to the method described in Section 5.4.2 (Line 22) and then checks for each query whether the value is smaller than the price (Lines 22 to 24). If the query q_i has lower value than the calculated price, the utility of the query would drop by allocating more runtime. Consequently, the query will be removed from the set S (Line 24) and it will not be considered for further allocations. If an index is removed from S, the prices have to be recalculated, as the corresponding query is not anymore available to share the prices. The loop continues to calculate new prices and remove indices until all the remaining queries q_i for $i \in S$ have positive utility by getting more runtime allocated and consequently, there are no more indices to be removed from S (Line 25).

The next step is to calculate for each query q_i the value τ, which indicates how much runtime can be allocated at most to the query. τ is the minimum of the remaining time limit T_i and the remaining budget b_i divided by the price p_i (Line 27). These values get updated at each iteration. Next, $\tau_{\min}$ gets updated, which is the smallest value of τ encountered during the iterations (Line 27). $\tau_{\min}$ is the runtime that will be allocated to each query in S, eventually. The queries get only the smallest overall value allocated because after the time period $\tau_{\min}$ at least one query will drop out and the prices have to be recalculated. Finally, the allocated runtime t_i, the time limit T_i, the payment π_i, and the budget b_i are updated to reflect the allocated time (Lines 28 to 29). If the remaining runtime limit T_i or budget b_i is exhausted, the query q_i will not be considered for further runtime allocations, as this would violate the queries budget and/or runtime limit, and the index i is removed from S (Line 30).

In the following, we provide an example for Algorithm 5.1:

Example 1. *In this example, we use the queries and prices from Figure 5.2. For this, we assume that the three queries q_1, q_2, and q_3 have value coefficients $v_1 = 4$, $v_2 = 20$, $v_3 = 10$, budgets $b_1 = 10$, $b_2 = 31$, $b_3 = 20$, and runtime limits $T_1 = 2$, $T_2 = 5$, $T_3 = 2.5$.*

After initializing the values, the algorithm defines $S = \{1, 2, 3\}$. Afterwards, the algorithm calculates the prices which are depicted in Figure 5.2: $p_1 = 5.5$, $p_2 = 11$, $p_3 = 4.5$. As $p_1 > v_1$, the algorithm removes the first query from S. This leaves the algorithm with $S = \{2, 3\}$. As S has changed, the prices have to be recalculated, this time without q_1. This leads to the following price shares: $p_2 = 15.5 < v_2$, $p_3 = 4.5 < v_3$. Next, the algorithm calculates $t_{\min} = 2 = \frac{b_2}{p_2}$. It allocates this runtime to q_2 and q_3 and update budget, allocated runtime, runtime limit, and price: $b_2 = 0$, $b_3 = 11$, $t_2 = 2$, $t_3 = 2$, $T_2 = 3$, $T_3 = 0.5$, $\pi_2 = 31$, $\pi_3 = 9$. As the budget of q_2 reached zero, the query gets removed from S.

In the second iteration of the algorithm, S only contains the index 3. The new price share for q_3 is $p_3 = 7 < 10$. This time, $t_{\min} = 0.5 = T_3$, as the remaining runtime limit is smaller than the remaining budget divided by the cost. As before, budget, allocated runtime, runtime limit, and price get updated: $b_3 = 7.5$, $t_3 = 2.5$, $T_3 = 0$, $\pi_3 = 12.5$. As q_3 reaches $T_3 = 0$, it also gets removed from S and S becomes empty and hence, the algorithm terminates. The algorithm returns for each query the allocated runtime and total price: $t_1 = 0$, $t_2 = 2$, $t_3 = 2.5$, $\pi_1 = 0$, $\pi_2 = 31$, $\pi_3 = 12.5$.

Algorithm 5.1 Allocation and pricing algorithm

Data: Value coefficients $v_1, \ldots, v_n$, budgets $b_1, \ldots, b_n$, time limits $T_1, \ldots, T_n$
Result: Allocated runtimes $t_1, \ldots, t_n$, total prices $\pi_1, \ldots, \pi_n$

$t_1, \ldots, t_n \leftarrow 0$ $\pi_1, \ldots, \pi_n \leftarrow 0$ $S \leftarrow \{1, \ldots, n\}$ **while** $S \neq \emptyset$ **do**
  do
    $p_1, \ldots, p_n \leftarrow$ `calculatePriceShares(`S`)` **for** $i \in S$ **do**
      if $v_i < p_i$ **then**
        $S \leftarrow S \setminus \{i\}$
  while *A query has been removed from* S
  $\tau_{\min} \leftarrow +\infty$ **for** $i \in S$ **do**
    $\tau \leftarrow \min\left(\frac{b_i}{p_i}, T_i\right)$ $\tau_{\min} \leftarrow \min(\tau_{\min}, \tau)$
  for $i \in S$ **do**
    $t_i \leftarrow t_i + \tau_{\min}$ $T_i \leftarrow T_i - \tau_{\min}$ $\pi_i \leftarrow b_i + \tau_{\min} \cdot p_i$ $b_i \leftarrow b_i - \tau_{\min} \cdot p_i$ **if** $b_i \leq 0$ or $T_i \leq 0$ **then**
      $S \leftarrow S \setminus \{i\}$
return $t_1, \ldots, t_n$, $\pi_1, \ldots, \pi_n$

As discussed before, `calculatePriceShares` runs in $O(n)$ time, where n is the number of queries. Each for loop in Algorithm 5.1 runs at most n times, because $|S| \leq n$. The outer while loop also runs at most n times because at each iteration, at least one element is removed from S. Finally, also the inner while loop runs at most n times because the loops stops when no more element from S are removed and there are at most n elements which can be removed from S. This means that Algorithm 5.1 has a runtime complexity of $O(n^2)$. The space complexity is $O(n)$ as the algorithm has to keep track of a constant number of values for each query.

5.4.4 Properties of the Cost Sharing Algorithm

We now study to which degree Algorithm 5.1 fulfils the requirements.

Requirement R1. To see that each consumer benefits from participation (R1), note that our price sharing method guarantees that a consumer shares prices only from those operators and sources relevant for the query. Hence, the assigned price share can never be higher than the price of running the query in isolation, in which case the isolated query has to cover the whole price alone. In addition, if there is any overlap in common sources or operators with other queries, the assigned price share will be strictly cheaper than running the query in isolation. Next, it is important to note that Algorithm 5.1 allocates runtime and charges payments as long as the assigned price share is smaller or equal than the consumer's value. This means that the consumer's utility is maximized, given the information provided, because as long as the value is still higher or equal than the price, any additional runtime increases the overall utility by the difference between the

value and the price share. In addition, the algorithm never allocates a runtime to a query which would exceed the runtime limit or would cause to surpass the budget. Since the consumer's utility is maximized given the assigned price shares, and the assigned price shares are smaller (or equal if there is no price sharing possible), the consumer's utility must be smaller (or equal) than running the query in isolation. Hence, each consumer profits from our model.

Requirement R2. A crucial part to fulfil Requirement R1 is the assumption that each consumer provides value, runtime limit, and budget, truthfully. However, this might not be the case, necessarily. In order that the assumption can be considered realistic, our model has to make sure that a consumer cannot gain utility by misreporting about his or her utility. As our model guarantees to maximize the consumers' utilities given the assigned price shares, a consumer can only profit from misreporting if this yields a lower price assignment. While our price sharing algorithm takes into account the parameters given by the consumer, these parameters are only used to determine if, and for how long a given query is executed. Using equal need cost sharing guarantees that the price shares themselves are independent of these parameters. Hence, the price assignments cannot be influenced by misreporting and, consequently, no consumer can benefit from this.

Requirement R3. We have just shown that a consumer cannot benefit from manipulation by misreporting value, runtime limit, or budget. The remaining opportunity for an illegitimate utility gain would be trough manipulating the query itself. The question is whether a consumer can gain utility by modifying the query. Requirement R3 is satisfied under the following conditions:

Condition 1 (No partial value)**.** *A consumer has no value for a partial stream. A partial stream is the result of running the query only on a subset of all the relevant operators and sources.*

Condition 2 (No external collaboration)**.** *Different consumers cannot collaborate externally, that is, accessing data together as one consumer and then splitting the streaming data and payments among each other outside of the model.*

Condition 1 ensures that a consumer has zero value for the streaming result if any of the relevant sources or operators is removed from the model. This condition is important because the model assumes that the declared value of the consumer reflects the consumers true valuation of the received data stream. If a consumer had a value for partial streams, the model would have missing information on the value on this partial stream. Hence, the model would not be able to decide whether the current setting would indeed maximize the customer's utility. Therefore, the customer would have an opportunity to gain utility by deviating from the original query such that only the partial stream is queried. An extension to our model where a customer can provide multiple valuations for multiple (partial) streams could, however, circumvent this problem. However, this would add another layer of complexity because different valuations for the partial stream would require a multi-objective optimization. This is beyond the scope of this paper.

Condition 2 ensures that multiple consumers cannot collaborate outside the model. If this would be possible, a group of consumers could act as one consumer inside our model

and reduce the assigned price share, potentially. This is because our algorithm for the price distribution calculates the price shares based on the different queries present in the model. A special kind of license attached to the streaming result of the query could, for example, prohibit the redistribution of the queries outside the legal entity (a person or company) which bought the data. Again, this is beyond the scope of our paper, however.

Under these two conditions, we can now show that Requirement R3 is also fulfilled by our model: the only way to benefit by deviating from the query is by introducing additional sources and operators, or, by removing sources and operators and doing part of the access/computation locally. Introducing additional sources and operators does not benefit the consumer, as these additional nodes do not change the price distribution of existing sources and operators. Hence, introducing such additional nodes only increase the associated price share for a query. Removing sources and operators does not yield a reduced price share, either. This is because with each source or operator which is not shared with other queries, the consumer has to compensate the full price instead of only part of it. As a result, the consumer has no incentive to deviate from the original query.

5.5 Evaluation

The aim of the evaluation is to study the runtime behavior and financial advantages of using our price sharing model. The runtime of our algorithm is a crucial metric because the price shares needs to be updated whenever a new query arrives in the system. The total number of participating queries is limited by how fast our algorithm can integrate new queries by recalculating the price shares. The more queries are participating in the price sharing, the more potential money savings for each participant.

For the evaluation, we created randomly generated graphs, as described below. We used an artificial setting because we wanted to measure the behavior of our model under varying degree of overlap with otherwise equal conditions. No real world dataset would allow such a fine grained control of the desired parameter.

We used a fixed model for the continuous queries where each query consists of 7 operator nodes and 8 source nodes. Each operator has two inputs and one output. The computational nodes are arranged in 3 layers (Figure 5.3). In total there are 500 sources available in our scenario.

To build the scenario for our evaluation, we create a number of n different queries. Each newly added query has for each operator node an *overlap probability* p that a particular node coincides with an existing node belonging to the same layer. If two nodes coincide, their child nodes in subsequent layers do as well. In addition, we created 500 source nodes from which each operator node in layer 4 randomly chooses 2. The number of sources is fixed and does not grow with the number of queries in the evaluation. Table 5.1 shows the total number of nodes (including sources) for different number of queries and different overlap probability p.

For our evaluation, we focused on the budget as the constraining parameter. Since all three parameters, value coefficient, time limit, and budget act as a constraint on the same outcome variable, the allocated runtime, it is sufficient to study the influence of one of

Table 5.1: Number of nodes for different number of queries and different overlap probability p.

Queries	$p = 0$	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$	$p = 0.6$	$p = 0.7$	$p = 0.8$	$p = 0.9$	$p = 1$
1	507	507	507	507	507	507	507	507	507	507	507
10	570	566	554	540	545	535	528	526	523	517	516
100	1200	1114	1012	898	873	828	732	706	652	623	606
1000	7500	6493	5583	4942	4180	3516	2904	2475	2102	1780	1506
10000	70500	60831	51977	44146	37270	30436	24859	20154	16062	13011	10506

them. In our setting, each operator node and each source have a price of 1. The value coefficient and the time limit is set to infinite for each query. This means that the budget is the only constraining parameter. The budget for each query is a uniformly randomly assigned value between 0 and 1, which prevents different queries having the exact same allocated runtime, a situation that would be very unrealistic and would improve the runtime of the allocation algorithm heavily.

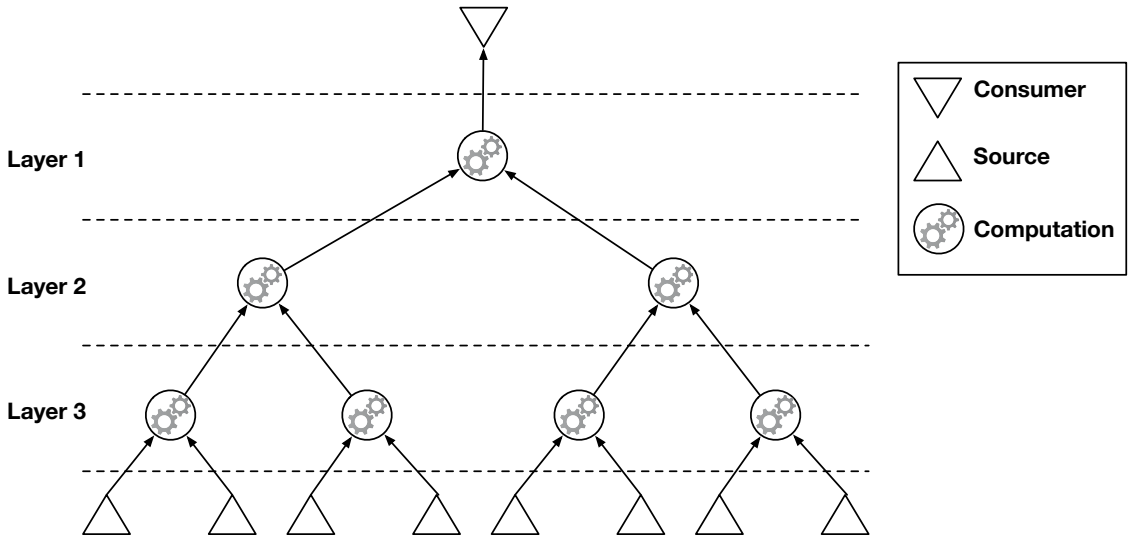


Fig. 5.3: A query with 7 operator nodes and 8 source nodes.

For the purpose of this evaluation, the algorithm was implemented in Java and executed on a quad-core 2.8 GHz i7 processor, 16GB of RAM. Figure 5.4 shows the time needed to do the calculation of the price sharing and the allocated runtime. The top graph in Figure 5.4 shows how the calculation time varies for different probabilities and over different number of queries. As we can see, the calculation time does not vary much when varying the overlap probability p but does vary much when varying the number of queries. The bottom graph shows how much runtime a query gets allocated on average for different overlap probabilities and different number of queries. Note that a query gets more allocated runtime the more overlap the queries have and the more queries are present in our model. The reason for this is that more overlap clearly means more sharing between

queries and hence, more money savings for each query. The higher the number of queries, the higher is the positive influence of overlapping computation, as there are more queries which share prices.

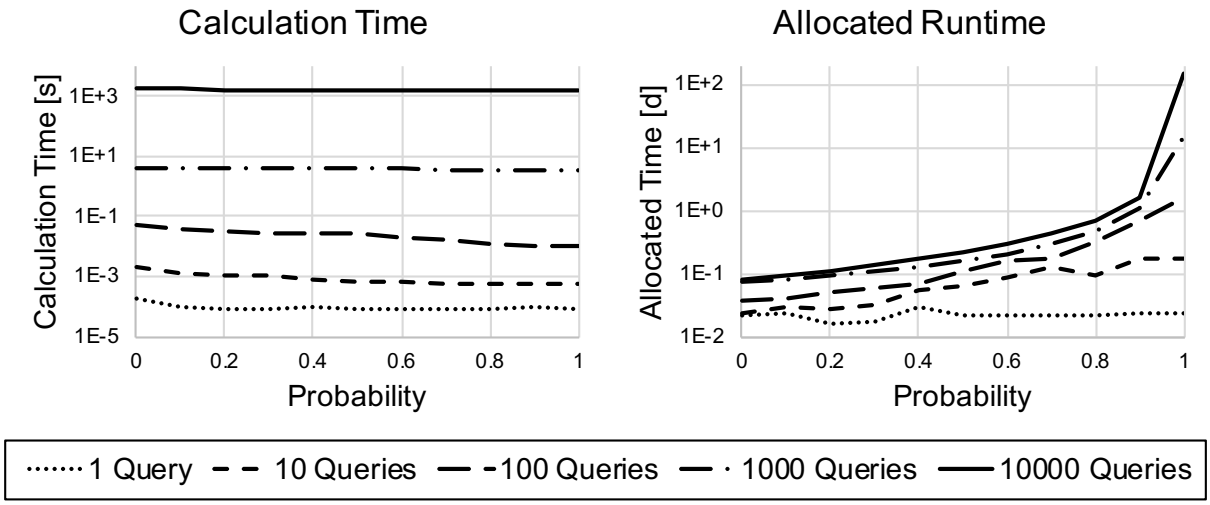


Fig. 5.4: Calculation time and allocated runtime.

5.6 Limitations and Conclusion

The growing interest in streaming data creates new opportunities for the WoD to save money by sharing computational fees. Inspired by the easy data integration of the WoD, we studied scenarios where different continuous queries share the price of common operations and common sources. As we have seen, consumers can profit from price sharing in stream processing settings. Our proposed method allows calculating payments which ensure participation and do not encourage manipulation.

Our method is somewhat limited in its scalability when tens of thousands of queries are simultaneously involved in our algorithm. Whilst this is a very large number that will first have to be reached in practical environments, we do foresee solutions to this limitation: first, queries can be partitioned into smaller groups of only a few thousands of queries and the price sharing can be done for each such partition individually. Second, one can try to relax some of the requirements we imposed on the price sharing in favor of better runtime of the algorithm. Third, one might find approximating algorithms with reasonable properties. Another possible drawback could be found in the assumptions. We suggested that the pricing of the raw data is exogenous/given: in current scenarios this seems reasonable, but we will have to be explore the pricing of the raw data in future work.

Whatever the limitations, our paper signifies a first step in exploring an extremely important problem in RDF Stream processing: how to fund the data production and

computation. We believe that the requirements we stated in our paper are an important aspect of price sharing in the streaming setting and that every solution which aims at implementing price sharing in this setting has to address those requirements. As far as we know, our paper is the first to introduce a price sharing algorithm that both adhere to these requirements and is feasible under realistic run-time constraints. As such, our contribution paves the way for research on establishing realistic price sharing approaches for stream processing.

Part III

Appendix

List of Figures

List of Tables

List of Code Listings

List of Algorithms

References

Acharya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: Join synopses for approximate query answering. In: Proc. of the 1999 ACM SIGMOD International Conference on Management of Data. pp. 275–286 (1999)

Acosta, M., Vidal, M.E., Lampo, T., Castillo, J., Ruckhaus, E.: Anapsid: An adaptive query processing engine for sparql endpoints. In: International Semantic Web Conference (1). pp. 18–34 (2011)

Aggarwal, G., Feldman, J., Muthukrishnan, S.: Bidding to the top: Vcg and equilibria of position-based auctions. In: Approximation and Online Algorithms. WAOA 2006 (2007)

Aggarwal, G., Feldman, J., Muthukrishnan, S., Pál, M.: Sponsored search auctions with markovian users. In: WINE '08. pp. 621–628 (2008)

Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: EC '06 Proceedings of the 7th ACM conference on Electronic commerce. pp. 1–7 (2006)

Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: WWW. pp. 635–644. ACM (2011)

Aumann, R.J., Shapley, L.S.: Values of Non-Atomic Games. Princeton University Press, Princeton, New Jersey (1974)

Auyoung, A., Grit, L., Wiener, J., Wilkes, J.: Service contracts and aggregate utility functions. In: Proceedings of the IEEE Symposium on High Performance Distributed Computing. pp. 119–131 (2006)

Babock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings Sysmposium on Principles of Database Systems (2002)

Bakos, Y., Brynjolfsson, E.: Bundling information goods: Pricing, profits, and efficiency. Management Science 45(12), 1613–1630 (1999), URL http://EconPapers.repec.org/RePEc:inm:ormnsc:v:45:y:1999:i:12:p:1613-1630

Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: A continuous query language for RDF data streams. International Journal of Semantic Computing 04(01), 3–25 (2010)

Basca, C., Bernstein, A.: Querying a messy Web of Data with Avalanche. Journal of Web Semantics 26, 1–28 (2014)

Belson, D.: Akamai's [state of the internet]. Q3 2015 report. Tech. rep., Akamai Technologies (2015)

Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284(5), 34–43 (May 2001), URL http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

Bertrand, J.L.F.: Book review of theorie mathematique de la richesse sociale and of recherches sur les principles mathematiques de la theorie des richesses. Journal de Savants 67, 499–508 (1883)

Billera, L.J., Heath, D.C., Raanan, J.: Internal telephone billing rates–a novel application of non-atomic game theory. Operations Research 26(6), 956–965 (1978)

Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. In: Communications of the ACM. vol. 13, pp. 422–426 (1970)

Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M., Tang, Y.: On the false-positive rate of Bloom Filters. Information Processing Letters 108(4), 210–213 (2008)

Bublies, P.: Fuel of the future: Data is giving rise to a new economy. The Economist (May 6th) (2017)

Buil-Aranda, C., Arenas, M., Corcho, O., Polleres, A.: Federating queries in sparql 1.1: Syntax, semantics and evaluation. Web Semantics: Science, Services and Agents on the World Wide Web 18(1), 1–17 (2013a)

Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: SPARQL web-querying infrastructure: Ready for action? In: et al., A.H. (ed.) The Semantic Web – ISWC 2013. vol. 8219, pp. 227–293 (2013b)

Cailliau, R.: A little history of the world wide web. https://www.w3.org/History.html (1995)

Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: Proceedings of the International Semantic Web Conference ISWC '10. pp. 96–111 (2010)

Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.: Monitoring streams: a new class of data management applications. In: VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases. pp. 215–226 (2002)

Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: Approximate query processing using Wavelets. In: The VLDB Journal. vol. 10, pp. 199–223 (2001)

Clarke, E.H.: Multipart Pricing of Public Goods. Public Choice 2, 19–33 (1971)

Craswell, N., Zoeter, O., Taylor, M., Ramsey, B.: An experimental comparison of click position-bias models. In: WSDM '08 Proceedings of the 2008 International Conference on Web Search and Data Mining. pp. 87–94 (2008)

Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Computing Surveys 44(3), 15:1–15:62 (2012)

Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (February 2014)

Dash, D., Kantere, V., Ailamaki, A.: An economic model for self-tuned cloud caching. In: Proceedings of the 2009 IEEE International Conference on Data Engineering (2009)

Deep, S., Koutris, P.: QIRANA: A framework for scalable query pricing. In: SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data. pp. 699–713 (2017)

Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science 1(1–2), 59–83 (2017)

Dustdar, S., Pichler, R., Savenkov, V., Truong, H.L.: Quality-aware service-oriented data integration: Requirements, state of the art and open challenges. In: ACM SIGMOD Record, vol. 41, pp. 11–19. ACM New York, NY, USA (2012)

Edelman, B., Ostrovsky, M.: Strategic bidder behavior in sponsored search auctions. In: Decision Support Systems. vol. 43, pp. 192–198 (2007)

Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second-price auction: Selling billions of dollars. The American Economic Review 97(1), 242–259 (2007)

Erling, O., Mikhailov, I.: RDF support in the Virtuoso DBMS. Tech. rep., OpenLink Software (Apr 2009)

Feigenbaum, L., Williams, G.T., Clark, K.G., Torres, E.: Sparql 1.1 protocol. https://www.w3.org/TR/sparql11-protocol/ (March 2013)

Glott, R., Schmidt, P., Ghosh, R.: Wikipedia survey – overview of results. Tech. rep., United Nations University MERIT (March 2010)

Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: Proceedings of the 2nd International Workshop on Consuming Linked Data. vol. 782, pp. 13–24. Bonn, Germany (2011), URL http://uni-koblenz.de/~goerlitz/publications/GoerlitzAndStaab_COLD2011.pdf

Groves, T.: Incentives in Teams. Econometrica 41(4), 617–631 (1973)

Grubenmann, T., Bernstein, A., Moor, D., Seuken, S.: Challenges of source selection in the WoD. In: Proceedings of the International Semantic Web Conference ISWC '17 (2017a)

Grubenmann, T., Dell'Aglio, D., Bernstein, A., Moor, D., Seuken, S.: Decentralizing the Semantic Web: who will pay to realize it? In: Proceedings of the Workshop on Decentralizing the Semantic Web (DeSemWeb) (2017b), URL http://ceur-ws.org/Vol-1934/contribution-01.pdf

Guo, F., Liu, C., Kannan, A., Minka, T., Taylor, M., Wang, Y.M., Faloutsos, C.: Click chain model in web search. In: WWW '09 Proceedings of the 18th international conference on World wide web. pp. 11–20 (2009)

Hamilton, J.: The cost of latency. Perspectives (October 31 2009), URL http://perspectives.mvdirona.com/2009/10/the-cost-of-latency/

Harris, S., Seaborne, A.: SPARQL 1.1 query language. https://www.w3.org/TR/sparql11-query/ (March 2013)

Harth, A., Umbrich, J., Hogan, A., Decker, S.: Yars2: a federated repository for querying graph structured data from the web. 6th International Semantic Web Conference (ISWC) pp. 211–224 (2007)

Hartig, O., Bizer, C., Freytag, J.C.: Executing SPARQL queries over the web of Linked Data. 8th International Semantic Web Conference ISWC2009 pp. 293–309 (2009)

Hose, K., Schenkel, R.: Towards benefit-based RDF source selection for SPARQL queries. In: SWIM '12 Proceedings of the 4th International Workshop on Semantic Web Information Management. Scottsdale, Arizona (2012)

Ioannidis, Y.E., Christodoulakis, S.: On the propagation of errors in the size of join results. In: Proceedings of ACM SIGMOD Conference. pp. 268–277 (1991)

Ioannidis, Y.E., Poosala, V.: Histogram-based approximation of set-valued query answers. In: Proceedings of the 25th International Conference on Very Large Data Bases. pp. 174–185 (1999)

Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Mining and Knowledge Discovery 18(1), 140–181 (2009)

Kossmann, D.: The state of the art in distributed query processing. ACM Computing Surveys 32(4), 422–469 (2000)

Koutris, P., Upadhyaya, P., Balazinska, M., Howe, B., Suciu, D.: Toward practical query pricing with querymarket. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. pp. 613–624 (2013)

Labrinidis, A., Qu, H., Xu, J.: Quality contracts for real-time enterprises. In: Proceedings of the 1st international conference on Business intelligence for the real-time enterprises. pp. 143–156. BIRTE'06, Springer-Verlag, Berlin, Heidelberg (2007), URL http://dl.acm.org/citation.cfm?id=1761128.1761140

Lai, K., Rasmusson, L., Adar, E., Zhang, L., Huberman, B.A.: Tycoon: An Implementation of a Distributed, Market-based Resource Allocation System. Multiagent and Grid Systems 1(3), 169–182 (August 2005), URL http://dl.acm.org/citation.cfm?id=1233813.1233816

Langegger, A., Wöß, W., Blöchl, M.: A semantic web middleware for virtual data integration on the web. Lecture Notes In Computer Science (2008)

Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Proceedings of the International Semantic Web Conference ISWC '11. pp. 370–388 (2011)

Lipton, R.J., Naughton, J.F., Schneider, D.A.: Practical selectivity estimation through adaptive sampling. In: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data. pp. 1–11 (1990)

Lohr, S.: For impatient web users, an eye blink is just too long to wait. New York Times (February 29 2012), URL http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html

Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Boston, MA (2001)

Malone, T.W., Fikes, R.E., Howard, M.T.: Enterprise : a Market-like Task Scheduler for Distributed Computing Environments (Nov 1983), URL http://dspace.mit.edu/handle/1721.1/47750

Margara, A., Urbani, J., van Harmelen, F.: Streaming the web: Reasoning over dynamic data. Web Semantics: Science, Services and Agents on the World Wide Web 25 (2014)

Mauri, A., Calbimonte, J.P., Dell'Aglio, D., Balduini, M., Brambilla, M., Della Valle, E., Aberer, K.: TripleWave: Spreading RDF Streams on the Web. In: International Semantic Web Conference (2). Lecture Notes in Computer Science, vol. 9982, pp. 140–149. Springer (2016)

Microsoft Corporation: Microsoft White Paper: Windows Azure Marketplace. http://go.microsoft.com/fwlink/?LinkID=268648&clcid=0x0409 (2011)

Moor, D., Grubenmann, T., Seuken, S., Bernstein, A.: A double auction for querying the web of data. In: The Third Conference on Auctions, Market Mechanisms and Their Applications (2015)

Moor, D., Seuken, S., Grubenmann, T., Bernstein, A.: Core-selecting payment rules for combinatorial auctions with uncertain availability of goods. In: Twenty-Fifth International Joint Conference on Artificial Intelligence. pp. 424 – 432 (2016)

Moulin, H., Laigret, F.: Equal-need sharing of a network under connectivity constraints. Games and Economic Behavior 72, 314–320 (2011)

Moulin, H., Shenker, S.: Serial cost sharing. Econometrica 60(5), 1009–1037 (1992)

Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. Journal of Artificial Intelligence Research 29, 19–47 (2007)

Özçep, Ö.L., Möller, R., Neuenstadt, C.: A Stream-Temporal Query Language for Ontology Based Data Access. In: KI. Lecture Notes in Computer Science, vol. 8736, pp. 183–194. Springer (2014)

Ozsu, T., Valduriez, P.: Principles of Distributed Database Systems (2nd Edition). Prentice Hall, 2 edn. (1999), URL http://www.citeulike.org/user/zflavio/article/379597

Peitz, M., Waldfogel, J.: The Oxford Handbook of the Digital Economy. erausgegeben von Oxford University (2012)

Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Transactions on Database Systems (TODS) 34(3) (2009)

Prasser, F., Kemper, A., Kuhn, K.A.: Efficient distributed query processing for autonomous RDF databases. In: Proceedings of the 15th International Conference on Extending Database Technology - EDBT '12. pp. 372–383 (2012)

Puiu, D., Barnaghi, P., Tönjes, R., Kümper, D., Ali, M.I., Mileo, A., Parreira, J.X., Fischer, M., Kolozali, S., Farajidavar, N., Gao, F., Iggena, T., Pham, T.L., Nechifor, C.S., Puschmann, D., Fernandes, J.: Citypulse: Large scale data analytics framework for smart cities. In: IEEE Access. vol. 4, pp. 1086–1108 (2016)

Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. Proceedings of the 5th European semantic web conference on The semantic web: research and applications pp. 524–538 (2008)

Richardson, M., Dominowska, E., Ragno, R.: Predicting clicks: Estimating the click-through rate for new ads. In: WWW '07 Proceedings of the 16th international conference on World Wide Web. pp. 521–530 (2007)

Rinne, M., Nuutila, E., Törmä, S.: INSTANS: high-performance event processing with standard RDF and SPARQL. In: Proceedings of the ISWC 2012 Posters & Demonstrations Track (2012)

Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngonga Ngomo, A.C.: A fine-grained evaluation of SPARQL endpoint federation systems. Semantic Web Journal 7, 493–518 (2016), URL http://svn.aksw.org/papers/2014/fedeval-swj/public.pdf

Saleem, M., Ngonga Ngomo, A.C.: HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In: The Semantic Web: Trends and Challenges. pp. 176–191 (2014)

Saleem, M., Ngonga Ngomo, A.C., Parreira, J.X., Deus, H.F., Hauswirth, M.: DAW: Duplicate-AWare federated query processing over the Web of Data. In: et al., A.H. (ed.) The Semantic Web – ISWC 2013. Lecture Notes in Computer Science, vol. 8218, pp. 574–590. Springer, Berlin, Heidelberg (2013)

Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: A benchmark suite for federated semantic data query processing. International Semantic Web Conference pp. 585–600 (2011)

Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization techniques for federated query processing on Linked Data. In: Proceedings of the 10th International Semantic Web Conference. pp. 601–616 (2011a)

Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization techniques for federated query processing on linked data. In: International Semantic Web Conference (1). pp. 601–616 (2011b)

Shapley, L.S.: A value for n-person games. In: Contributions to the Theory of Games II, Annals of Mathematical Studies, vol. 28. Princeton University Press (1953)

Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys 22(3), 183–236 (Sep 1990)

Stonebraker, M., Aoki, P.M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., Yu, A.: Mariposa: A wide-area distributed database system. VLDB J. 5(1), 48–63 (1996)

Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. ACM SIGMOD Record 34(4), 42–47 (2005)

Sutherland, I.E.: A futures market in computer time. Commun. ACM 11(6), 449–451 (Jun 1968), URL http://doi.acm.org/10.1145/363347.363396

Taelman, R., Heyvaert, P., Verborgh, R., Mannens, E.: Querying Dynamic Datasources with Continuously Mapped Sensor Data. In: International Semantic Web Conference (Posters & Demos). CEUR Workshop Proceedings, vol. 1690. CEUR-WS.org (2016)

Tallevi-Diotallevi, S., Kotoulas, S., Foschini, L., Lécué, F., Corradi, A.: Real-time urban monitoring in Dublin using semantic and stream technologies. In: Proceedings of the International Semantic Web Conference ISWC '13. pp. 178–194 (2013)

Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres, A.: Comparing data summaries for processing live queries over Linked Data. World Wide Web 14(5), 495–544 (2011)

Van Alstyne, M., Brynjolfsson, E., Madnick, S.: Why not one big database? principles for data ownership. Decis. Support Syst. 15(4), 267–284 (Dec 1995), URL http://dx.doi.org/10.1016/0167-9236(94)00042-4

Varian, H.R., Harris, C.: The vcg auction in theory and practice. American Economic Review 104(5), 442–45 (2014)

Vickrey, W.: Counterspeculation, Auctions, and Competitive Sealed Tenders. The Journal of Finance 16(1), 8–37 (1961)

Vidal, M.E., Castillo, S., Acosta, M.: On the selection of SPARQL endpoints to efficiently execute federated SPARQL queries. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XXVII. pp. 109–149 (2016)

Wagner, A., Speiser, S., Harth, A.: Semantic web technologies for a smart energy grid: Requirements and challenges. In: Proceedings of the International Semantic Web Conference ISWC '10. pp. 33–37 (2010)

Waldspurger, C.A., Hogg, T., Huberman, B.A., Kephart, J.O., Stornetta, W.S.: Spawn: a Distributed Computational Economy. IEEE Transactions on Software Engineering 18(2), 103–117 (1992), URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=121753

Wang, X., Zheng, Z., Wu, F., Dong, X., Tang, S., Chen, G.: Strategy-proof data auctions with negative externalities. In: Proceedings of the International Conference on Autonomous Agents Multiagent Systems (AAMAS). pp. 1269–1270 (2016)

Wilkens, C.A., Cavallo, R., Niazadeh, R.: Gsp – the cinderella of mechanism design. In: WWW '17 Proceedings of the 26th International Conference on World Wide Web. pp. 25–32 (2017)

Zemánek, J., Schenk, S., Svátek, V.: Optimizing SPARQL queries over disparate RDF data sources through distributed semi-joins. 7th International Semantic Web Conference (ISWC) (Oct 2007)

Zhu, Z.A., Chen, W., Minka, T., Zhu, C., Chen, Z.: A novel click model and its applications to online advertising. In: WSDM '10 Proceedings of the third ACM international conference on Web search and data mining. pp. 321–330 (2010)

Zollinger, M., Basca, C., Bernstein, A.: Market-based sparql brokerage with matrix: Towards a mechanism for economic welfare growth and incentives for free data provision in the web of data. Tech. Rep. IFI-2013.4 (2013)

Curriculum Vitæ

Tobias Peter Grubenmann

Tüfenacker 24
LI-9488 Schellenberg
Fürstentum Liechtenstein

Born on February 27 1986 in Grabs SG, Switzerland

Education

09/2014 – ongoing PhD Student in Computer Science, University of Zurich

- Research topic: Monetization strategies for the Semantic Web
- Supervision: Master theses, Bachelor theses, and software projects
- Teaching: Teaching Assistant for the courses "Informatics and the Economy" and "Semantic Web Engineering"

02/2013 – 08/2014 MSc in Computational Science, University of Zurich

09/2008 – 02/2013 BSc in Mathematics with Minor in Philosophy, University of Zurich

Professional Experience

07/2018 – ongoing Postdoctorial Fellow at The University of Hong Kong

05/2016 – 08/2016 Software Engineering Intern at Google Inc.

07/2013 – 08/2013 Trainee at Zurich Insurance Group

07/2012 – 10/2012 Trainee at Zurich Insurance Group

Publications

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Financing the Web of Data with Delayed-Answer Auctions. In *WWW 2018: The 2018 Web Conference*, 2018

Tobias Grubenmann. Monetization Strategies for the Web of Data. In *WWW '18 Companion: The 2018 Web Conference Companion*, 2018

Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Challenges of source selection in the WoD. In *Proceedings of the International Semantic Web Conference ISWC '17*, 2017

Tobias Grubenmann, Daniele Dell'Aglio, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Decentralizing the Semantic Web: who will pay to realize it? In *Proceedings of the Workshop on Decentralizing the Semantic Web (DeSemWeb)*, 2017

Dmitry Moor, Sven Seuken, Tobias Grubenmann, and Abraham Bernstein. Core-selecting payment rules for combinatorial auctions with uncertain availability of goods. In *Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016

Dmitry Moor, Tobias Grubenmann, Sven Seuken, and Abraham Bernstein. A double auction for querying the web of data. In *The Third Conference on Auctions, Market Mechanisms and Their Applications*, 2015

Participation in Boards and Panels

11/2017 – 06/2018 Representative of the PhD students at the Senate of the University of Zurich

01/2016 – 06/2018 Representative of the PhD students at the Faculty Board of the Faculty of Economics, Business Administration and Information Technology

02/2018 Program Committee member of the Web Stream Processing Workshop at WWW 2018 Satellites

Zurich, Tuesday 11th September, 2018